Escuela Politécnica Superior

Navegación y conducción autónoma de vehículos con geometría Ackermann

Grado en Ingeniería Robótica

Trabajo Fin de Grado

Autor:
Roberto Saúl Cova Rocamora

Tutor:
Fernando Torres Medina

Universitat d'Alacant
Universidad de Alicante

Julio 2019

Navegación y conducción autónoma de vehículos con geometría Ackermann

Autor

Roberto Saúl Cova Rocamora

Tutor

Fernando Torres Medina

Dpto. de Física, Ingeniería de Sistemas y Teoría de la Señal

Grado en Ingeniería Robótica

Escuela Politécnica Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Julio 2019

Dedicado a David,
mi hermano.

Agradecimientos

En primer lugar, quiero agradecerle todo su apoyo y consejo a mi tutor, Fernando Torres Medina, sin el cual este trabajo no habría sido posible. Asimismo agradecer su ayuda a Pablo Gil y Francisco Andrés Candela Herías, docentes del grupo de investigación AUROVA. A los doctorandos Iván del Pino y Miguel Ángel Muñoz y al estudiante de máster Miguel Ángel Contreras, por dedicarme su tiempo y esfuerzo durante tanto tiempo cuando estaba dando mis primeros pasos en la robótica móvil y por todos aquellos ratos que pasamos desarrollando BLUE y CLEAR.

En segundo lugar, me gustaría agradecer este trabajo a los profesores y profesoras del Grado de Ingeniería Robótica de la Universidad de Alicante. Gracias a ellos hoy tengo los conocimientos que tengo.

Y por último, quiero agradecerles a mis padres, a mis hermanos, al resto de mi familia y a mi pareja su apoyo y su ánimo en los buenos y en los malos momentos.

¡Muchas gracias a todos!

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación y Justificación

En este trabajo se presenta el estado del arte de las técnicas más relevantes en la conducción autónoma ya que pretende ser una guía donde se aúnen los conocimientos básicos para el estudio de la conducción autónoma y del desarrollo de un vehículo terrestre no tripulado (*Unmanned Ground Vehicle* en inglés) en el cual poder probar, desarrollar e investigar en este área del conocimiento.

Se va a tratar el problema de la conducción desde dos perspectivas. Por un lado, la percepción del entorno, centrándose en la parte visual. Y por otro lado, en los algoritmos de control del vehículo. Se tratarán tanto métodos clásicos como los que están todavía en investigación. Desde los algoritmos básicos de control, hasta técnicas tan novedosas como el Aprendizaje Profundo (o *Deep Learning*) ya sea para percepción del entorno, como para el control del vehículo.

También se realiza una explicación detallada de las técnicas y de los resultados obtenidos en la experimentación, dejando el pseudocódigo en los anexos y recursos tan importante como son bases de datos (entre otros) en la bibliografía, sobre todo para el aprendizaje automático.

1.2. Estructura

El trabajo está organizado en **6 capítulos** y **2 anexos**, siguiendo el libro de estilo de la Escuela Politécnica Superior de la Universidad de Alicante. Se ha definido la organización de este trabajo tratando de explicar los 2 aspectos más importantes dentro de los vehículos autónomos, tratando de explicarlos desde mayor a menor nivel de abstracción de las tareas.

En el **capítulo 2** se aborda la una introducción a las conducción autónoma definiendo aspectos tan importantes como los niveles de conducción autónoma que existen, los elementos tanto tangibles como intangibles que los vehículos suelen utilizar según investigaciones recientes, así como evaluar los riesgos y a seguridad como un aspecto crucial en la navegación autónoma. En el **capítulo 3** se aborda la percepción de los vehículos autónomos, centrándose en la percepción visual a través de cámaras estereoscópicas, así como los métodos de Segmentación de Imagen aplicados a la conducción autónoma. El **capítulo 4** trata la tarea de menor nivel dentro de los coches autónomos, el control de velocidad y dirección. Por último el **capítulo 5** trata sobre el desarrollo y creación de un controlador de bajo nivel utilizado para robotizar plataformas reales con geometría Ackermann. En el **capítulo 6** capítulo se describen los experimentos realizados, sus resultados. Por último se describen las conclusiones de este trabajo y los trabajos futuros.

1.3. Publicaciones

El presente Trabajo de Fin de Grado, ha sido desarrollado como consecuencia de la investigación realizada en el laboratorio AUROVA (AUtomática RObótica y Visión Artificial) de la Universidad de Alicante. El resultado de dicha investigación queda patente en cuatro publicaciones. De las cuales, una de ellas ha sido publicada en una revista científica (JCR). Otras dos han sido publicadas en congresos internacionales. Y

la cuarta en un congreso nacional.

Presenting BLUE: A robot for localization in unstructured environments [del Pino et al., 2018a] ha sido publicado en *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. Este artículo hace una primera introducción al robot BLUE, como una necesidad en el mundo de la investigación de plataformas reales donde experimentar en robótica móvil. Mi contribución en este artíclo se ha basado en ayudar al desarrollo de la arquitectura hardware y software, del módulo de control, así como de la elección del hardware.

Speed Estimation for Control of an Unmanned Ground Vehicle using Extremely Low Resolution Sensors [del Pino et al., 2018b] ha sido publicado en *15th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*. Este artículo trata la problemática de controlar una plataforma Ackermann con sensores de baja resolución para medir la velocidad aplicando un Filtro de Kalman Unidimensional. Mi participación en este proyecto ha sido, ayudar a la experimentación de todo el proceso, determinando la necesidad de utilizar un estimador, hasta aplicar dicho estimador y sus resultados.

Deeper in BLUE [del Pino et al., 2019] ha sido publicado en *Journal of Intelligent & Robotic Systems*. Este artículo trata en profundidad todos los aspectos relacionados con el robot BLUE, en el cual he contribuido con el desarrollo del módulo de control, así como en los aspectos básicos de diseño y localización GNSS con RTK del robot.

CLEAR. Un módulo para la robotización de máquinas Ackermann [Cova-Rocamora et al., 2018] en el congreso nacional *XXXIX Jornadas de Automática*. En este artículo se describe en profundidad la arquitectura y software y hardware de un módulo de control genérico para controlar plataformas de configuración Ackermann y su integración en el robot BLUE. Mi participación en este proyecto ha sido, ayudar al desarrollo de la arquitectura de CLEAR, su programación, pruebas e integración, así como a elegir los componentes necesarios para crearlo.

Capítulo 2

Introducción a la conducción autónoma

La conducción humana es una tarea compleja que involucra muchos aspectos diferentes que se interrelacionan para lograr llegar con seguridad al destino deseado. Debido a la alta complejidad de esta tarea, se puede dividir en tres subtareas interrelacionadas entre sí:

1. **Planificación**, toma de decisiones y localización: Para ello se debe diferenciar el espacio de tiempo al que se va a referenciar. Por ejemplo, a largo plazo el objetivo es hallar el camino óptimo (o un subóptimo) que indique cómo ir desde una posición inicial a un destino deseado. A corto plazo, se tienen que tomar decisiones vitales para la conducción, como pueden ser cambios de carril, frenada de emergencia, velocidad deseada, etcétera. Y a plazo inmediato está el control de bajo nivel del vehículo, como seguir una curva, cuanto acelerar o frenar. Pero tomar decisiones no es una tarea fácil debido a la gran cantidad de reglas que se pueden aplicar a hora de conducir, el error en los instrumentos de medida y las situaciones imprevistas. Se tratará más en profundidad esta tarea en las capítulos 3 y 5.

2. **Percepción del entorno**: El vehículo ha de ser capaz de analizar el estado propio (posición, velocidad, aceleración, orientación y velocidad de rotación) y del entorno para poder decir qué hacer en cada momento. Para ello habrá que ser capaz de Detectar y Responder frente a Objetos y Eventos (*Object and Event detection and Response* u OEDR). Para ello se diferencia entre objetos estáticos (carreteras y señales viales) y objetos dinámicos (otros coches, peatones, ciclistas, etc). La tarea de percepción se ve dificultada por la iluminación, las condiciones atmosféricas, las oclusiones de objetos e incluso del propio ruido de los sensores, así como de otros factores. Se tratará más en profundidad esta tarea en la capítulo 4.

3. **Control del vehículo**: Hay que tener en cuenta que el movimiento tradicional de un vehículo responde a dos controles diferentes. Por un lado el control longitudinal o de velocidad que se encarga de acelerar y frenar. Por otro lado el control lateral o de dirección que se encarga de controlar la dirección del vehículo. Además las restricciones que suponen los vehículos (al poseer una configuración *Ackermann* y, por tanto, no ser holonómicos) influyen en los tipos de controladores. Aunque a priori pueda parecer que velocidad y dirección son sistemas aislados, hay que tener en cuenta que (a parte de condiciones externas al coche como puede ser la geometría del carril) la velocidad va a influir directamente en la dirección. Se tratará más en profundidad esta tarea en la capítulo 6.

En este trabajo se han dividido estas subtareas en 5 módulos distintos (ver Figura 2.1). Dichas subtareas de la tarea de conducción son compartidas tanto por los agentes humanos como si se trata de coches autónomos. Pero focalizando en los sistemas de conducción autónoma hay que tener en cuenta el Dominio de Diseño Operativo del agente u *Operational Design Domain* (ODD) [Czarnecki, 2018] en inglés. El ODD define las condiciones bajo las cuales va a ser capaz de trabajar el sistema. Dentro de él, habrá que especificar si va a ser capaz de funcionar durante todo el día o solo a ciertas horas, si puede funcionar en cualquier situación atmosférica o hay que limitarlas, el tipo de carretera, etcétera. Como se puede ver, el ODD abarca un amplio rango de variables, es

por eso que una buena definición previa de este es crucial para la conducción autónoma, ya que con él se podrán determinar las condiciones para las cuales el vehículo funcionará de forma segura.

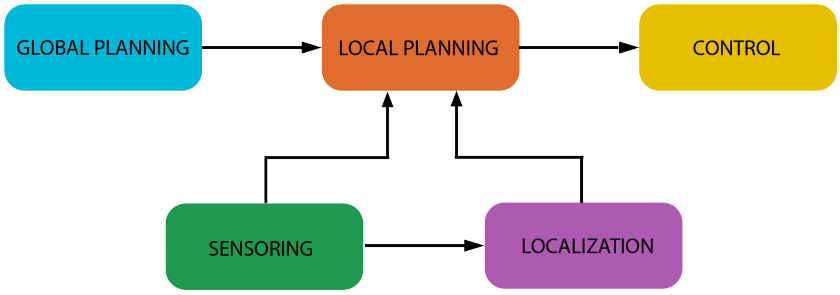


Figura 2.1: Módulos básicos de un vehículo autónomo.

2.1. Niveles de autonomía

Una vez establecido el ODD del vehículo, es importante tener claro el nivel de autonomía que tendrá. Esto deriva a la siguiente cuestión "¿Cómo clasificar los niveles de autonomía de un vehículo?". Una buena clasificación debería contemplar al menos las tres subtareas de conducción (así como otras tareas realizadas por los humanos como poner los intermitentes), la atención requerida por los humanos que estén dentro, y si se requieren acciones por parte de los humanos.

De hecho, ya existe una clasificación en niveles para vehículos autónomos llevada a cabo por primera vez en 2014 por la la *Society of Automotive Engineers* [SAE, 2018] y que se ha ido desarrollando con el tiempo. Esta clasificación propone cinco niveles de autonomía (ver Figura 2.2):

- **Nivel 0**: No hay ninguna autonomía. Los humanos realizan toda la tarea de conducción o casi toda.
- **Nivel 1**: Asistente de conducción. El coche es capaz de realizar un Control de Crucero Adaptativo o *Adaptative Cruise Control*. O el coche es capaz de mante-

nerse en su carril. Por tanto, en este nivel el coche puede realizar un control lateral del vehículo (dirección) o longitudinal (aceleración y frenado) pero no ambos a la vez.

- **Nivel 2**: Conducción parcialmente autónoma. El coche es capaz de realizar el control de velocidad y dirección simultáneamente.
- **Nivel 3**: Conducción autónoma condicional. Además de todo lo anterior, el vehículo es capaz de Detectar y Responder a Objetos y Eventos (*Object and Event detection and Response* u OEDR).
- **Nivel 4**: Conducción de alta autonomía. Además de todo lo anterior, el vehículo es capaz de responder en situaciones de emergencia.
- **Nivel 5**: Conducción totalmente autónoma. El coche es capaz de conducir en cualquier ODD dando una respuesta segura ante cualquier situación o emergencia.

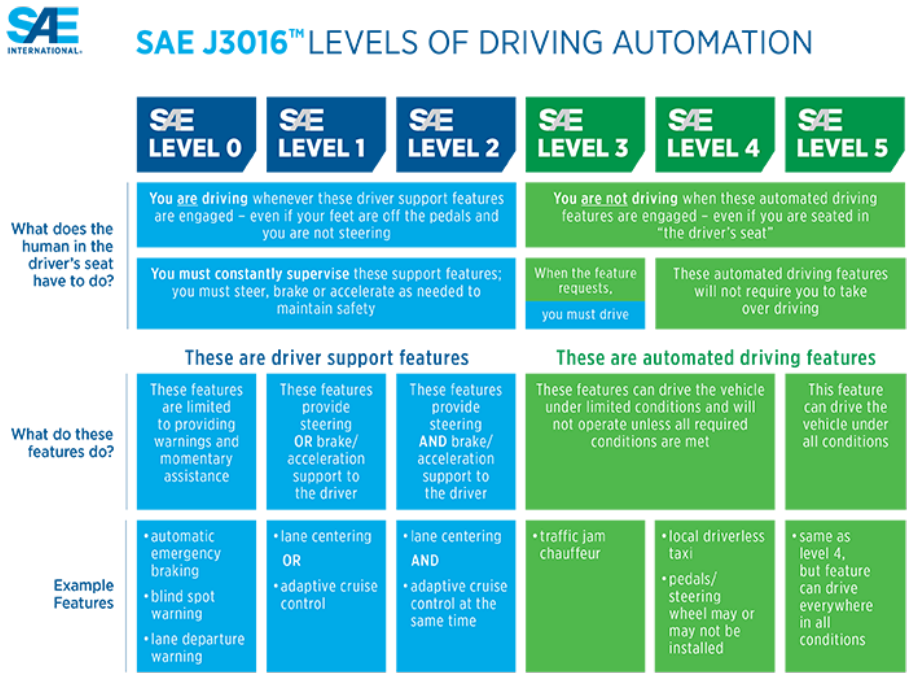


Figura 2.2: Niveles de autonomía en la conducción. Fuente: https://www.sae.org/

Aunque esta clasificación está ampliamente extendida, está limitada debido al hecho de

que habría que tener en cuenta que dos coches distintos, con ODD distintos, entrarían en el mismo nivel de autonomía y esto podría llevar a situaciones de alto riesgo.

2.2. Sensores y Hardware Básico

Los sensores son una parte esencial del vehículo autónomo, ya que tienen un papel fundamental en la subtarea de la percepción. Estos, permiten conocer el estado del vehículo así como el mundo que le rodea.

Estos, se pueden clasificar en dos grandes categorías: Los propioceptivos, que proporcionan información sobre la velocidad, aceleración, dirección, entre otras variables importantes propias del vehículo. Y los exteroceptivos, que captarán la información del mundo rodea al coche, estos servirán para detectar la carretera, peatones, crear un mapa, etcétera. De hecho, estos últimos son esenciales ya que además de ser necesarios para percibir el mundo, permiten estimar algunas variables internas del coche, usando técnicas como Odometría Visual (*Visual Odometry*), aunque esta estimación es más ruidosa que leyendo el valor directamente de un sensor propioceptivo. Una vez se ha realizado una clasificación para los sensores, habría que entender las diferentes subtareas que se realizan en la conducción para poder tener una base sólida sobre la cual cimentar cuales serán los sensores básicos que necesitará todo vehículo autónomo que circule *on-road*. Dichos sensores serán:

- **Global Navigation Satellite Systems (GNSS)**: Permite conocer la posición y velocidad del vehículo.
- **Inertial Measurement Units (IMU)**: Este sensor proporciona información sobre la aceleración lineal y angular, de las cuales se puede derivar la velocidad y posición del vehículo.
- **Odometría**: Da información de la velocidad y dirección del vehículo a través de codificadores (*encoders*) en los motores y otras técnicas.

- **Cámaras**: Las cámaras proporcionan información fundamental sobre la escena. La visión monocular permite detectar objetos, obstáculos e información esencial. Y con cámaras estereoscópicas se obtiene información de profundidad correspondiente a la escena.
- **Lidar**: Proporciona información 3D detallada de la escena. También son muy utilizados los lidar 2D, sobre todo para mapeado.
- **Radar**: Permiten detectar de forma robusta objetos cuando nos encontramos en situaciones de poca visibilidad.
- **Ultrasonidos**: Muy utilizados hoy en día, sobre todo para aparcamiento autónomo debido a que suelen ser baratos y sencillos de usar.

Como podemos observar, para la conducción autónoma se usa un amplio rango de sensores para poder obtener toda la información posible de distintas fuentes. Esto nos servirá para fusionar las medidas usando filtros y así poder obtener datos fiables del estado del vehículo, así como crear un mapa y localizarse en él (SLAM) entre muchas otras cosas.

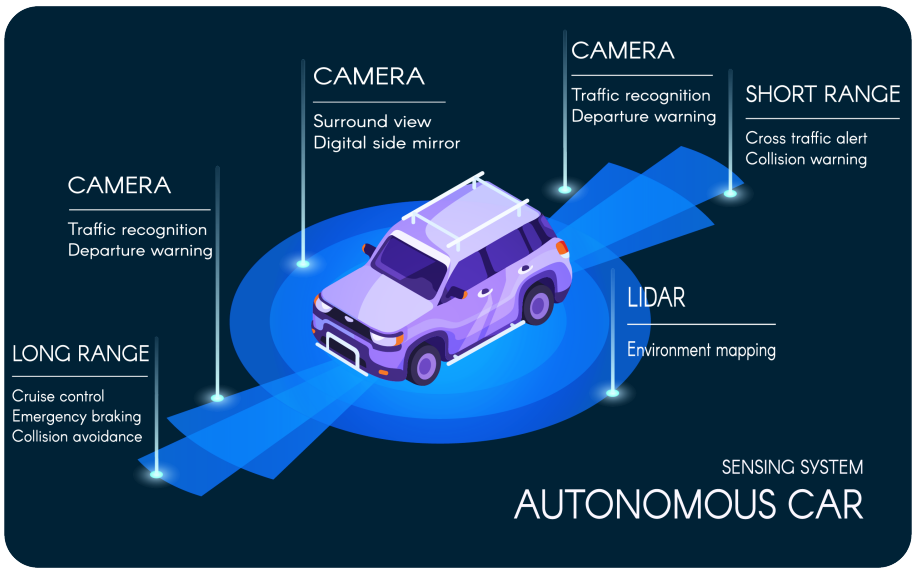


Figura 2.3: Sensores básicos para la conducción autónoma.

Una vez establecidos los sensores básicos, la siguiente tarea sería como distribuirlos por el vehículo. Para ello hay que tener en cuenta diferentes escenarios y condiciones, como por ejemplo conducir en autovía o en ciudad (terreno urbano), que permitan el control lateral y longitudinal, aparcamiento, tener en cuenta los puntos ciegos de otros sensores, entre otros. Basándose en la literatura actual y en el trabajo de Bussemaker [Bussemaker, 2014], una buena aproximación para la configuración de los sensores exteroceptivos sería la representada en la Figura 2.3.

Si bien es cierto que hay fabricantes de coches autónomos como Tesla y su Autopilot [Tesla,] que están apostando por eliminar sensores como el Lidar, y solamente usar cámaras, radar y ultrasonidos como sensores exteroceptivos. Aunque, su configuración es muy similar a la propuesta en la Figura 2.3. Por otro lado, cabe mencionar que debido a los altos requerimientos y velocidad de respuesta de un sistema autónomo, el uso de Unidades de Procesamiento Gráfico (GPUs) y Unidades Centrales de Procesamiento (CPUs) de última generación está muy extendido. Aunque su estudio está fuera del ámbito de este trabajo.

2.3. Arquitecturas Software

Una vez se ha definido la arquitectura hardware básica y sus componentes, el siguiente paso es definir cuál es la arquitectura software de alto nivel. Realmente dicha arquitectura no será muy distinta de la usada comúnmente en los robots móviles. Sin entrar mucho en detalle, cualquier arquitectura para el control de un vehículo autónomo debería tener los siguientes módulos (ver Figura 2.4):

- **Percepción del entorno**: Será capaz de obtener el estado del vehículo y detectar los objetos estáticos y dinámicos.
- **Mapeado del entorno**: Creará uno o varios mapas del entorno (Occupancy Grid, Localization Map y/o Detailed Road Map)

- **Planificación de movimiento**: Planificará la trayectoria a seguir así como variaciones en esta debido a obstáculos, restricciones, etc.
- **Controlador**: Obtendrá un valor de dirección, aceleración y de freno.
- **Sistema de supervisión**: Supervisa cada uno de los módulos anteriores, así como los actuadores y los sensores.

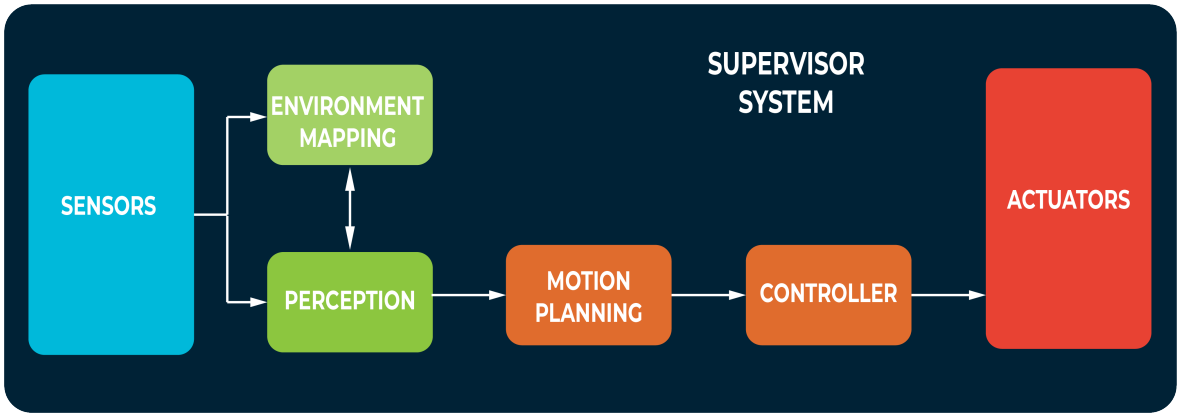


Figura 2.4: Arquitectura software básica.

Otras arquitectura similar es la que usó el equipo VictorTango en el DARPA Urban Challenge [Reinholtz, 2007].

2.4. Seguridad

Si en cualquier sistema autónomo la seguridad es importante, cuando están implicados coches que conducen de forma autónoma lo es aún más. Esto es debido a la gran variedad de escenarios a los que se tiene que enfrentar y las altas velocidades a las que puede circular poniendo en riesgo vidas humanas y daños materiales. Es por eso, que se necesita sistemas de supervisión y seguridad que supervisen en todo momento el estado del vehículo y cada una de las salidas y procesos de cada bloque (ver Figura 2.4).

Pero para entender cómo prevenir los casos de fallo y cómo actuar frente a ellos hay que saber cuales son. Se pueden clasificar en dos grandes grupos, los comunes a todos los vehículos (como pueden ser mecánicos, eléctricos, hardware o software) y lo que son más propios de coches autónomos (planificación, percepción, ciberseguridad, comunicaciones y la intensificación del riesgo de hardware y software al haber más componentes y líneas de código).

Pero hay situaciones externas al propio vehículo que pueden incrementar el riesgo de accidente, como por ejemplo las condiciones medioambientales. No es lo mismo conducir de noche que de día, ni con lluvia o con nieve.

Hay arquitecturas genéricas (que no solo se usan para la automoción) para ayudar a priorizar las causas de fallo de un sistema. Cómo el Análisis de Árboles de Fallas o "Fault Trees" que se basan en los árboles de probabilidad y la lógica booleana para determinar la probabilidad de que se produzca un fallo por distintas razones. Desde el nodo raíz se va expandiendo un árbol no binario donde cada nodo es un posible caso de fallo y cada hijo una de las causas de fallo del padre hasta llegar a los nodos hoja. Este método permite discriminar los casos de riesgo según una mayor probabilidad de que ocurran. Por contra, se necesita conocer esta probabilidad a priori, lo cual no siempre es posible y, por tanto, estos sistemas suelen basarse en estadística para encontrar las probabilidades de los nodos.

La arquitectura "Failure Modes and Effects Analyses" [Ben-Daya, 2009] se basa en dos conceptos: categorizar los posibles modos de fallo por prioridad, y eliminar o reducir dichos fallos comenzando por los más prioritarios. Para ello utilizan tres variables: *Severidad* del fallo. *Ocurrencia*, es decir, la frecuencia con la que ocurre dicho fallo. Y *Detección*, cuán fácil es detectarlo. Con estas tres variables se calcula el RPN o Risk Priority Number.

$$RPN = S \cdot O \cdot D \tag{2.1}$$

También hay arquitecturas específicas para el automovilismo, como "Safety of the Intended Functionality" (SOTIF) [Wendorff, 2017]. Se basa en la caracterización de los riesgos que entrañan las limitaciones de los sensores, fallos en los algoritmos o un mal uso por parte de los humanos. SOTIF está diseñado para los tres primeros niveles de autonomía (del 0 al 2).

Hoy en día también se pueden encontrar diversas arquitecturas de seguridad mucho más especializadas de compañías privadas, como la que utiliza Waymo [Waymo,] o General Motors [Motors,], entre otros fabricantes.

El trabajo de Kalra y Paddock [Kalra and Paddock, 2016] demostró que, para determinar que un vehículo autónomo es seguro, debería recorrer alrededor de 8 billones de millas (billón americano). Para ello, estima que una flota de 100 vehículos, conduciendo sin parar, tardaría alrededor de 400 años en demostrarlo.

2.5. Simuladores

Como se menciona en la sección anterior, la seguridad es una elemento esencial para poder implantar la conducción autonoma a nivel global. Por ello, se hace necesario la utilización de simuladores, sobre todo en los primeros estadios de desarollo de los algoritmos.

Hoy en día se pueden encontrar diversos simuladores en Internet, de hecho, incluso videojuegos comerciales pueden ser utilizados como simuladores. Cómo por ejemplo Gran Theft Auto V [Richter et al., 2016] [Richter et al., 2018] o de código abierto como TORCS [Bernhard Wymann, 2014]. También se pueden encontrar simuladores más específicos para la investigación como el de Udacity [Udacity, 2016b] o CARLA [Dosovitskiy et al., 2017]. Todos ellos tienen ventajas y desventajas, pero para desarrollar un enfoque preciso y hacerlo lo más real posible hace falta un simulador con escenarios realistas y que proporcionen información a tiempo real de la conducción.

En este trabajo se ha utilizado el simulador CARLA, debido a que, como demostró Dosovitskiy [Dosovitskiy et al., 2017], este es uno de los simuladores más completos hoy en día.

Capítulo 3

Percepción

La percepción del entorno es una parte vital de la conducción autónoma. Es importante tanto en la localización del vehículo, como en la comprensión del mundo que le rodea para realizar una acción de control.

En este capítulo, se abordan los conceptos básicos relacionados con la percepción visual, es decir, con cámaras. Estos son, el reconocimiento de objetos y la segmentación de la imagen.

En cuanto a la visión, se pueden encontrar métodos tradicionales para el reconocimiento de objetos en imágenes, como el algoritmo de Viola-Jones [Viola and Jones, 2001], o *Histogram of Oriented Gradients* [Dalal and Triggs, 2005], entre otros. Pero no fue hasta la introducción del Deep Learning en 2012 con Alexnet [Krizhevsky et al., 2012], cuando se dio un salto cualitativo en la clasificación de imágenes y posteriormente en la detección de objetos y segmentación de imágenes.

Es por ello que en este trabajo se van a abordar los aspectos de la percepción visual modernos, basados en **Redes Neuronales Convolucionales** (CNN).

3.1. Reconocimiento y Detección de Objetos

Los vehículos autónomos no solo necesitan saber qué objetos hay dada una imagen de una cámara, también necesitan saber donde están situados. A este problema se le conoce como detección de objetos. Este campo está avanzando día a día debido a la investigación científica. De hecho, hay APIs, como la desarrollada para TensorFlow de detección de Objetos [TensorFlow,], entre otras.

La detección de objetos ayuda al vehículo a distinguir y clasificar los distintos obstáculos a los que se puede enfrentar. Ya sean estáticos (árboles, señales de tráfico, carreteras, etcétera) o dinámicos (peatones, vehículos, ciclistas, etcétera). Es por ello que una buena detección va a permitir dar una respuesta óptima ante las diversas situaciones. En este sentido, el *Deep Learning* ha supuesto un avance para este campo [Ren et al., 2015]. Las arquitecturas de detección se basan en utilizar una CNN como extractor de características de la imagen, se añaden los *boungind boxes*, con las capas de salida se clasifican los objetos y se ajustan los bounding boxes y después se filtra usando un filtro de *Non-Maximum Suppression* (ver Figura 3.1).

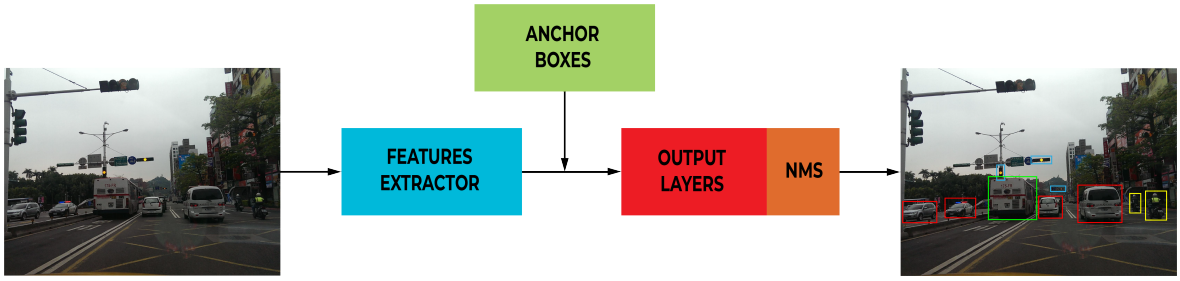


Figura 3.1: Ejemplo de Deep Learning para la detección de objetos.

La detección de objetos no siempre es perfecta. Puede pasar que algunos objetos no se detecten la imagen, o que se detecten de más. Una de las formas de filtrar las detecciones cuando se han detectado objetos de más es usando la segmentación de imagen.

3.2. Segmentación de Imagen

Como se ha mencionado al final de la sección anterior una buena segmentación puede ayudar a filtrar una mala detección. Para ello, se puede buscar dentro del área del *bounding box* proporcionado por el detector y descartar aquellas detecciones donde el número de píxeles de la clase sea menor a un cierto umbral. Dicho umbral dependerá del área total de la caja. Se puede asumir que si el área del objeto es menor a un cierto porcentaje del área de la caja, es una mala detección.

Pero la segmentación de imagen tiene más aplicaciones a parte del filtrado de detección de objetos. Quizás la aplicación más destacada de la segmentación de imagen en vehículos autónomos sea la detección del carril y del espacio por el cual se puede conducir.

Para esto también se utilizan arquitecturas basadas en CNNs. Las redes basadas en segmentación suelen llamarse encoder-decoder, ya que transforman una imagen en otra [Badrinarayanan et al., 2017]. Se basan en usar dos redes convolucionales, la primera, codifica la información extrayendo las características, y después una segunda red, decodifica la información transformándola en una imagen junto con la capa de salida (ver Figura 3.2).

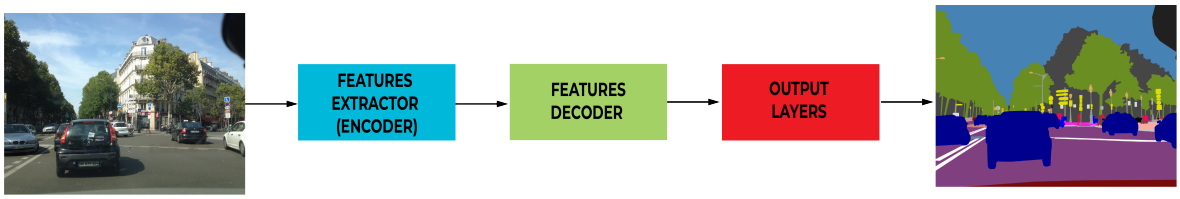


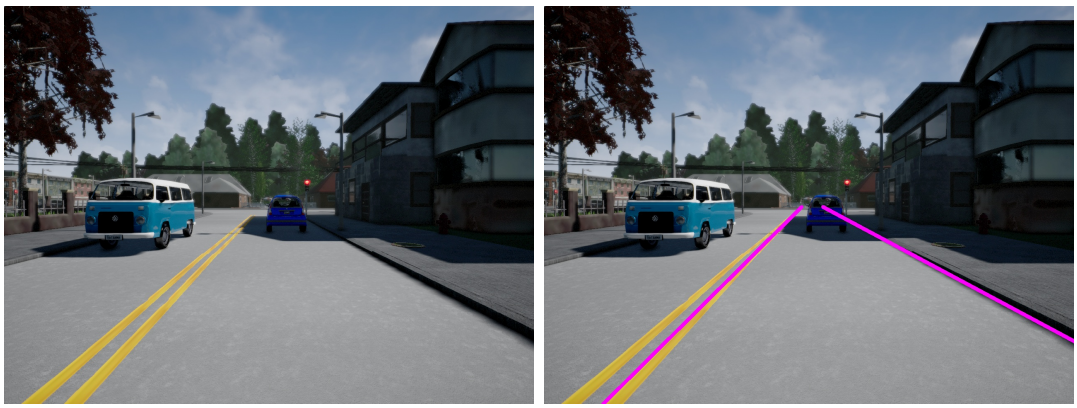
Figura 3.2: Ejemplo de Deep Learning para la segmentación de imágenes.

Por tanto, aunque ambas son muy similares la principal diferencia entre Segmentación y detección, es que la primera distingue a nivel de píxel en la imagen las distintas clases, mientras que la segunda proporciona una serie de cajas o regiones en la imagen que localizan las distintas clases u objetos.

3.3. Detección de Carril

La detección del carril es uno de los problemas básicos a la hora de la conducción autónoma. No basta simplemente con segmentar la carretera y las marcas viales, hace falta extraer más información para que se útil para la conducción. Datos como, la delimitación del carril actual e incluso la curvatura de la carretera es información esencial.

Una de las técnicas más comunes, es aplicar filtros de detección de bordes como Sobel, Prewwit o Canny a la imagen segmentada del carril. Y después aplicar algoritmos como la transformada de Hough para detectar las lineas que forman en carril (ver Figura 3.3).



(a) Imagen del Simulador CARLA (b) Imagen filtrada con detección de carril

Figura 3.3: Detección del carril usando Sobel y la transformada de Hough.

Uno de los problemas de Hough es que no puede identificar curvas. Esto es una limitación importante en la conducción, ya que las carreteras suelen tener curvatura en ciertos tramos. Y aunque es una buena base de partida, se pueden aplicar otros métodos que permiten detectar el carril y su curvatura.

En esta sección se presenta otro método para una detección más robusta del carril. Asumiendo que se comienza centrado en el carril y que la carretera es en un primer momento razonablemente recta (al igual que en la Figura 3.3) se puede aplicar el algoritmo anterior para hacer una primera detección del carril. Una vez hecho esto,

se puede aplicar un algoritmo rastree el carril y lo divida en una serie de puntos que definan la curva (ver Figura 3.4). He llamado a este algoritmo búsqueda por ventanas dinámicas ya que se crean ventanas de búsqueda que tratan de encontrar donde se encuentra el centro del carril en la ventana. Una vez definidos los puntos, trata de ajustar el polinomio de grado n (en este trabajo se utiliza un polinomio de segundo grado) que defina el carril. Al usar un polinomio de grado mayor que 1, se puede obtener la curvatura de la carretera, lo cual se puede implementar en el controlador para mejorar la conducción.

(a) Imagen de carril bici (b) Imagen filtrada con detección de carril

Figura 3.4: Detección del carril usando ventanas dinámicas.

Este algoritmo tiene una mayor fiabilidad en curva que el tradicional Hough. Sin embargo, no se puede garantizar su resultado cuando hay una segmentación deficiente o por ejemplo, cuando confunde marcas viales de señalización en carretera y las lineas del carril.

Capítulo 4

Control

Este capítulo trata de mostrar las diferentes estrategias que existen actualmente para controlar el vehículo. Esto no es ni más ni menos que reducir el error entre una consigna (posición, trayectoria, velocidad, etcétera) deseada, y el estado actual.

Para ello se desarrollan las dos estrategias básicas que existen hoy en día, el control clásico, el cual mediante leyes de control pre-programadas trata de realizar la conducción. Y las técnicas modernas, que utilizan sistemas de aprendizaje automático para dar respuesta al problema de la conducción.

4.1. Control Clásico

El control clásico se basa en separar la conducción en dos controladores distintos, por un lado el longitudinal o de velocidad, y el lateral o de dirección.

Para algunos controladores puede ser necesario conocer el modelado cinemático y dinámico del vehículo. (ver Apéndice A).

En este trabajo dichos controladores son genéricos y no dependen del modelo dinámico del vehículo (mientras sea Akcermann) ya que tratan de encubrir el modelo usando ga-

nancias. Aun así, es interesante conocer el modelado de un vehículo ya que se pueden desarrollar controladores más específicos. Para más información sobre otros controladores es interesante el trabajo de Snider [M. Snider, 2011].

4.1.1. Control Longitudinal

El control longitudinal [Rajamani, 2006] trata de eliminar el error relativo a una velocidad deseada dada la velocidad actual. Para ello transforma el error en velocidad en valores de aceleración y los transforma en un valor del pedal de aceleración o frenado para alcanzar dicha velocidad. Para ello se puede dividir el control en dos controladores, uno de alto nivel que obtendría el perfil de aceleración y otro de bajo nivel que transformaría ese valor en un porcentaje de posición de los pedales.

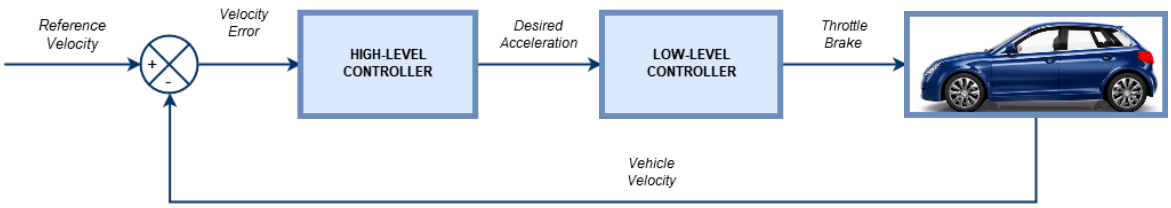


Figura 4.1: Esquema del ciclo de control longitudinal.

Un controlador muy usado para transformar velocidades en aceleraciones es un controlador Proporcional Integral y Derivativo (PID). Este sería el controlador de alto nivel del sistema.

$$\ddot{x}_{des} = K_p(\dot{x}_{ref} - \dot{x}) + K_I \int_0^t (\dot{x}_{ref} - \dot{x})dt + K_D \frac{d(\dot{x}_{ref} - \dot{x})}{dt} \tag{4.1}$$

Por otro lado, para el controlador de bajo nivel se necesita conocer el modelo dinámico del vehículo. Para ello se transforma la aceleración deseada en el par deseado en el motor y utilizando un "Engine Map" (ver Figura 4.2), se puede saber cual es la posición del acelerador deseada. Reorganizando la ecuación A.12 y sustituyendo de la ecuación A.10

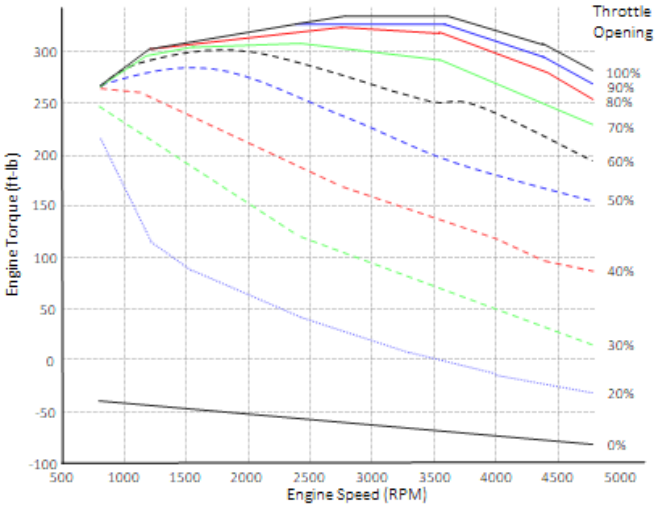


Figura 4.2: Ejemplo de un gráfico que relaciona el par del motor, las revoluciones y la posición del acelerador.

se llega a la siguiente expresión mediante la cual se puede obtener el par deseado en el motor.

$$T_e = T_{load} + \ddot{x}_{des} \cdot \frac{J_e}{r_{ef} \cdot GR} \tag{4.2}$$

En este trabajo se ha obviado el controlador de bajo nivel ya que se desconoce el modelo dinámico del vehículo con el que se trabaja en simulación. Por ello se han ajustado las ganancias del controlador PID para que compensen el error producido por este desconocimiento. Como se describe en el apartado de experimentación de este Capítulo, un PID que transforme de velocidad a valor del acelerador es suficiente.

4.1.2. Control Lateral

El control lateral busca eliminar el error relativo entre la trayectoria actual y la trayectoria deseada satisfaciendo las restricciones no holonómicas del vehículo. En este tipo de control se pueden encontrar dos tipos de controladores, los geométricos y los dinámicos. En este trabajo se va a tratar dos controladores geométricos: *Pure Pursuit* y *Stanley*. Y un controlador basado en espacio de imagen.

Para lo controladores geométricos hay que definir un término llamado "*Cross-track Error*". Este se define como la distancia desde un punto de referencia (eje trasero, delantero o centro de gravedad) hasta un punto cercano de la trayectoria deseada.

Pure Pursuit

El controlador Pure Pursuit o "seguimiento de zanahoria" [M. Snider, 2011]. Se basa en obtener el valor de dirección del coche a través de la curvatura del la trayectoria geométrica que hay entre el eje trasero del coche y el punto de referencia (ver Figura 4.3).

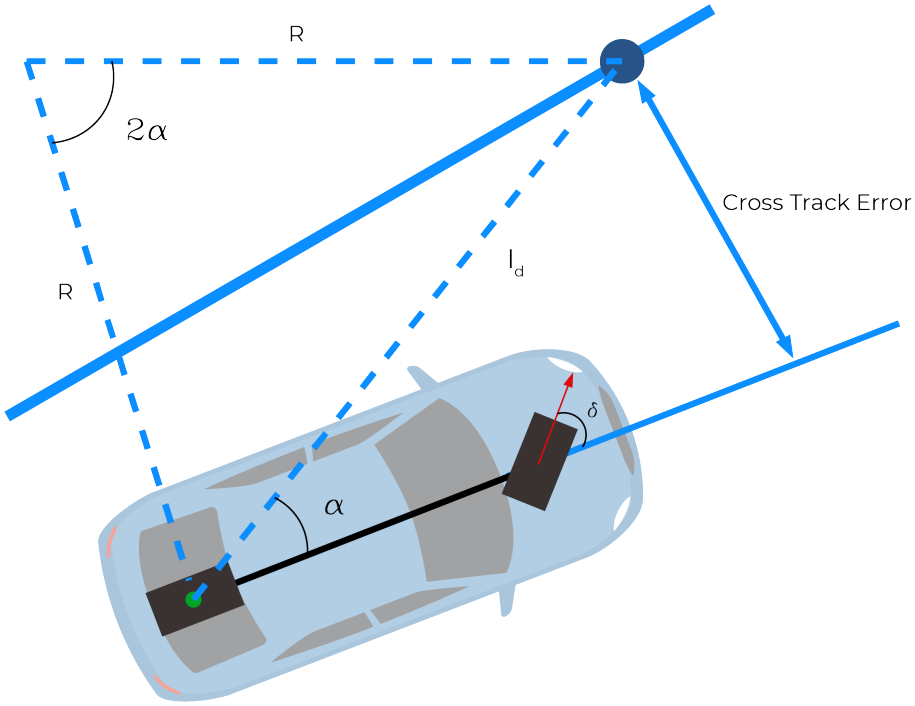


Figura 4.3: Esquema Pure Pursuit.

Usando la ley del seno:

$$\frac{l_d}{\sin 2\alpha} = \frac{R}{\sin\left(\frac{\pi}{2} - \alpha\right)} \tag{4.3}$$

$$\frac{l_d}{2\sin\alpha\cos\alpha} = \frac{R}{\cos\alpha} \tag{4.4}$$

Reorganizando:

$$R = \frac{l_d}{2\sin\alpha} \tag{4.5}$$

Por tanto, podemos obtener la curvatura:

$$k = \frac{1}{R} = \frac{2\sin\alpha}{l_d} \tag{4.6}$$

Para obtener el valor de α podemos usar el trigonometría básica usando el Cross-track error:

$$\sin\alpha = \frac{e}{l_d} \tag{4.7}$$

Y de ahí se puede calcular el valor de la dirección deseada:

$$\delta = \arctan\left(\frac{2L\sin\alpha}{l_d}\right) = \arctan\left(\frac{2Le}{l_d^2}\right), \quad \delta \in [\delta_{min}, \delta_{max}] \tag{4.8}$$

De la ecuación 4.8 se conocen todos los parámetros, por tanto, su resolución es trivial.

Ahora bien, este controlador usa una distancia deseada a la trayectoria fija, lo cual no es recomendable ya que no se necesita el mismo espacio para velocidades altas que cuando se circula a baja velocidad y hay más libertad de maniobra sin peligro. Como mejora se puede añadir una distancia dinámica en función de la velocidad:

$$l_d = K_d \cdot v \tag{4.9}$$

Donde K_d es una ganancia fijada de antemano.

Pero también se puede diseñar fijando una distancia mínima y máxima de búsqueda para suavizar el control con un interpolador y conociendo el rango de velocidades que se pueden alcanzar:

$$l_d = d_{min} + (d_{max} - d_{min}) \cdot \frac{v - v_{min}}{v_{max} - v_{min}} \tag{4.10}$$

Stanley

El controlador Stanley [Hoffmann et al., 2007] fue el usado por la universidad de Standford en el robot Stanley [Thrun et al., 2006] en el DARPA Grand Challenge. Se basa en dos principios, el "*cross-track error*" y en la alineación entre la trayectoria deseada y la del vehículo (ver Figura 4.4).

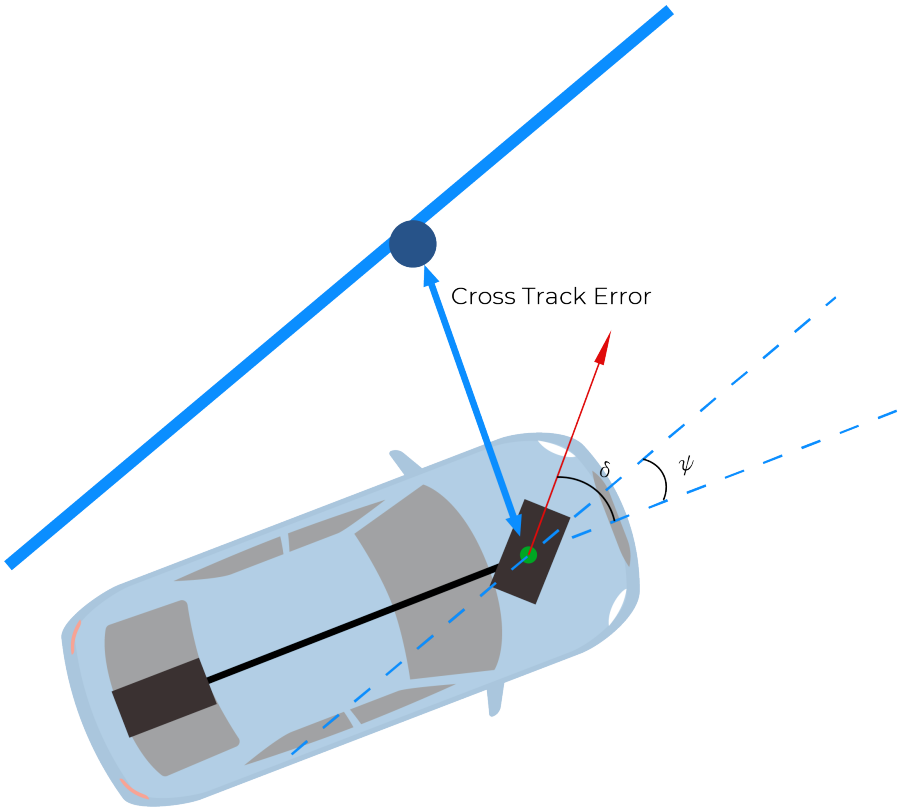


Figura 4.4: Esquema Stanley.

Este controlador usa como punto de referencia el eje delantero del vehículo. Es similar al controlador *Pure Pursuit*, salvo porque utiliza el punto de la trayectoria deseada más cercano a la posición actual, y porque tiene en cuenta la desviación angular (ψ) entre la trayectoria actual y la deseada.

Por tanto, la ley de control es la siguiente:

$$\delta(t) = \psi(t) + \arctan\left(\frac{k \cdot e(t)}{v}\right), \quad \delta(t) \in [\delta_{min}, \delta_{max}] \tag{4.11}$$

Este controlador es más sencillo y lo único que hay que ajustar es la ganancia K, ya que todos los demás términos son conocidos.

Basado en Imagen

A diferencia de *Pure Pursuit* y *Stanley*, el control basado en imagen no es geométrico, ya que no utiliza propiedades geométricas para obtener el ángulo de giro deseado.

Mientras que los controladores geométricos usan sistemas de coordenadas cartesianas (ya sea una referencia "mundo" o en función del robot), este controlador trabaja en coordenadas o espacio imagen. Esto permite trabajar de forma mucho más reactiva y local al espacio en el que se encuentre el robot, ya que no necesita un mapa definido, ni información 3D del entorno, la cual sueles ser costosa computacionalmente.

Trata de aplicar el mismo principio del controlador Longitudinal, pero en vez de usando un error en velocidad, se basa en el error en píxeles, la diferencia entre lo que está viendo el vehículo y lo que debería ver. Por ejemplo, el error en píxeles que hay hasta que el vehículo se centre en el carril.

Para la experimentación se ha usado la arquitectura propuesta en la Figura 4.5. En este caso se han utilizado comandos de alto nivel para mejorar el comportamiento en ciudad. Esta información de alto nivel es requerida ya que cada comando le indica al controlador si tiene que girar a la izquierda, derecha o seguir recto en la siguiente intersección.

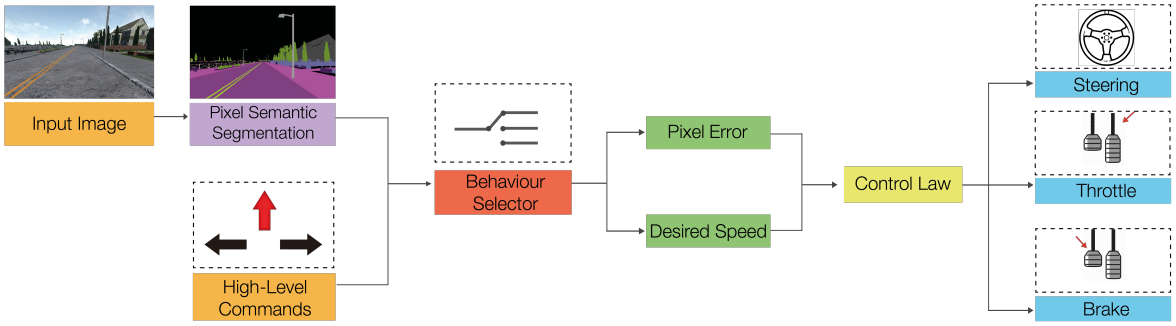


Figura 4.5: Esquema de control para un agente reactivo básico basado en navegación en espacio de imagen. Imágenes obtenidas del simulador CARLA.

4.2. Control Moderno

Las técnicas de control moderno están, hoy en día, en fase de investigación. Dichas técnicas buscan eliminar el controlador programado, por una sistema de aprendizaje que sea capaz de aprender la función del sistema que generalice lo suficiente para realizar una conducción óptima.

Dentro del control moderno hay dos vertientes, el *Deep Reinforcement Learning* [Yu et al., 2016] [Kendall et al., 2018] y el *End-to-End learning* [Bojarski et al., 2016] [Yang et al., 2018] o también llamado *Imitation Learning* [Codevilla et al., 2017]. Ambos métodos se basan en el Deep Learning para aprender. Usan la información en çrudo"de la cámara para obtener un valor de dirección y, depende del sistema, también de velocidad. Es por ello que muchas redes se nutren de información adicional como son la velocidad y dirección actual o información temporal.

4.2.1. Deep Reinforcement Learning

El Deep Reinforcement Learning (DRL) [Yu et al., 2016] [Kendall et al., 2018], o Aprendizaje por Refuerzo profundo, se basa en aplicar los algoritmos de aprendizaje

por refuerzo [Sutton and Barto, 1998] usando Deep Learning.

El aprendizaje por refuerzo se basa en, dado un estado actual (s) y una serie de acciones posibles (a), cual sería la acción que maximizaría la recompensa esperada en el futuro y, por tanto, sería mejor acción posible. Este algoritmo se puede implementar a través del llamado *Q-learning* [Watkins and Dayan, 1992]. La recompensa obtenida dado un estado y una acción se pueden calcular con la siguiente ecuación:

$$Q(s_{t+1}, a_{t+1}) = (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot max(Q(s_{t+1}, a_t)) - Q(s_t, a_t)) \quad (4.12)$$

Donde α es la constante de aprendizaje, s es el estado, a es una acción, r_t es la recompensa y γ es un factor de descuento.

Este algoritmo funciona cuando hay estados y acciones finitas, pero en el caso de la conducción se podría suponer como un sistema con infinitos estados y acciones posibles. Por ello, DRL utiliza Deep Learning para que le propio sistema aprenda la función que escoge cual es la acción que maximiza la recompensa esperada.

Existen diversas arquitecturas de DRL que han conseguido buenos resultados en la conducción autónoma cómo: Deep Q-learning [Yu et al., 2016] o basados en AC3 (*Asynchronous Actor-Critic Agent*) [Mnih et al., 2016]. Por otro lado, el trabajo de Kardell y Kousku [Kardell and Kuosku, 2017] hace una comparación entre Imitation Learning, ACER (*Actor-Critic with Experience Reply*) [Wang et al., 2016] y DDPG-Agent (*Deep Deterministic Policy Gradients*) en TORCS.

Uno de los problemas del aprendizaje por refuerzo, es que se necesitan muchos datos de la conducción para cuantificar cuanto de "mala.º "buena.está siendo la conducción. Por otro lado, es complicado llevar esto a la práctica en coches autónomos ya que al operar en un escenario real no tiene porque ser totalmente extrapolable.

4.2.2. Imitation Learning

El aprendizaje por imitación trata de enseñar a conducir al sistema a partir de ejemplos de conducción de un humano u otro sistema. Normalmente se usan para estimar la dirección, pero también se pueden usar para estimar la velocidad necesaria.

Están basados en el uso de Redes Neuronales Convolucionales (CNNs) e incluso Redes Neuronales Recurrentes (RNNs). Estas, necesitan una gran cantidad de datos para su entrenamiento y aprender a generalizar la función que optimiza la conducción. Esto implica recolectar horas de conducción humana experta, lo cual puede no estar al alcance de todo el mundo ni de todos los vehículos. De hecho para este trabajo se trató de adquirir los datos internos de un *Toyota Yaris* del 2015 usando el puerto *OBDII* del vehículo, pero mientras que los datos de posición del acelerador y velocidad, entre otros, eran transmitidos, la posición del volante no era transmitida. Esto deja de manifiesto la dificultad de obtener datos de conducción en un escenario real.

Hoy en día, se pueden encontrar diversos *datasets* públicos de conducción real, como el de Commaai [Commaai, 2016] y Udacity [Udacity, 2016a] entre otros. También se pueden encontrar *datasets* basados en simuladores, en concreto en CARLA [CARLA, 2017]. Uno de los problemas de los *datasets* es la incompletitud de la información, por ejemplo, no añaden meta-información de cuando va a haber un cambio de carril para poder enseñárselo a la red.

Capítulo 5

Desarrollo de una Plataforma Real

En este capítulo se describe el trabajo realizado para crear un módulo que facilite la robotización de vehículos con geometría Ackermann genérica. Este sistema permite el control y la monitorización del robot gracias a una interfaz hardware de bajo nivel con los actuadores y sensores del vehículo y a una interfaz software de alto nivel totalmente integrada en ROS (Robot Operating System). Dicho modulo recibe el nombre de CLEAR (Control Logic for Easy Ackermann Robotization) [Cova-Rocamora et al., 2018] y ha sido desarrollado para ser genérico y de código y hardware abiertos. Se pretende, por tanto, que este módulo sea una herramienta de ayuda a la investigación en coches autónomos para todo aquel que necesite transformar una plataforma móvil Ackermann en un vehículo autónomo. Finalmente, CLEAR ha sido integrado en una plataforma real.

5.1. CLEAR

CLEAR ha sido diseñado por una necesidad de trabajar y probar algoritmos de navegación autónoma en un escenario real. Ha servido para robotizar una carretilla industrial eléctrica Zallys Jespi Z015, la cual se está usando actualmente en investigación en tesis

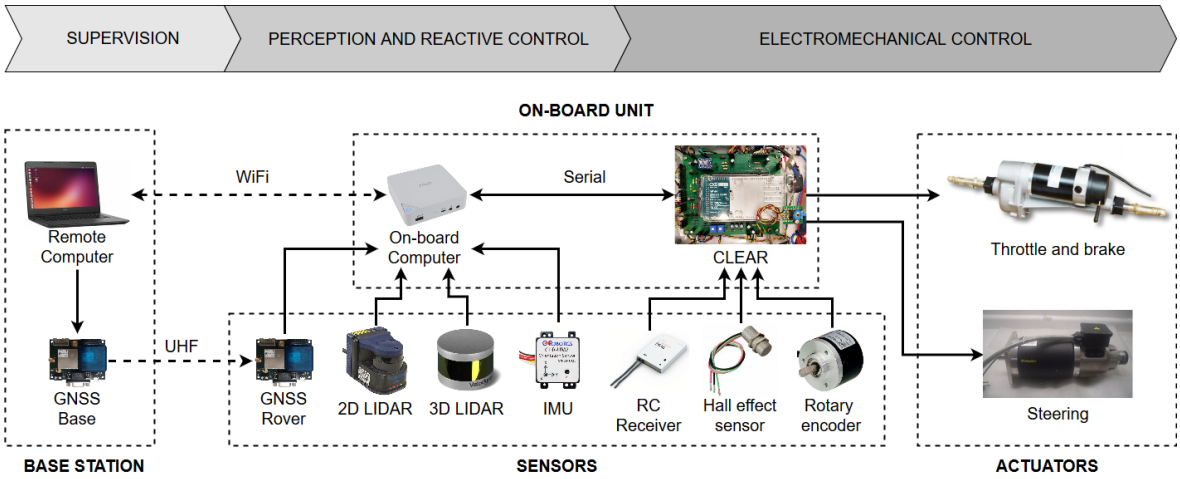


Figura 5.1: Arquitectura de CLEAR e integración con el vehículo.

doctorales con el nombre de BLUE (roBot for Localization in Unestructured Environments) [del Pino et al., 2018a] [del Pino et al., 2019]. La integración de ambos sistemas se puede ver en la Figura 5.1.

El objetivo de CLEAR es realizar un control robusto a bajo nivel del vehículo y servir de interfaz entre los algoritmos de conducción autónoma y el hardware de la plataforma. CLEAR puede ser utilizado para aplicaciones *on-road* y *off-road*, lo cual le permite gran escalabilidad.

Los aspectos básicos sobre los que se fomenta CLEAR son: hardware y software libres ya que tanto los códigos como los esquemas electrónicos están disponibles en el repositorio [AUROVA, 2018] de forma gratuita; incorporación de sistemas redundantes de seguridad para prevenir los fallos, para informar en caso de fallo y para actuar frente a estos; y por último facilitar al máximo la integración con los vehículos.

5.1.1. Software

El software del módulo ha sido diseñado para aportar genericidad y flexibilidad a la hora de ser exportado a distintas plataformas. Para ello se utiliza un lenguaje de alto nivel orientado a objetos, *C++*, que al ser compilado permite un menor tiempo de ciclo,

y por tanto, un control más rápido. Además, se ha integrado totalmente en ROS para poder visualizar y grabar todos los datos importantes del vehículo. Al ser un proyecto de código abierto, todo es modificable. Pero además, permite configurar la plataforma a través de los ficheros de configuración, facilitando la tarea de integración.

El proyecto está dividido en varias clases para encapsular cada uno de los componentes del vehículo. Esto permite propagar información entre las clases y poder visualizarla en ROS.

Máquina de Estados

CLEAR está implementado internamente como una máquina de estados que define cada uno de los modos operacionales en los que puede estar la plataforma:

- *Control por ROS*: En este modo, el robot se controla utilizando los *topics* de ROS. Por seguridad, es requiere una capa de seguridad reactiva de alto nivel para evitar colisiones. Esta información se recibe a través de ROS y que establece las velocidades máximas permitidas en función de la distancia a los obstáculos circundantes. Si no se reciben los *topics* de seguridad, el robot pasa al estado de Emergencia, desactivando los motores y activando los frenos.
- *Control Remoto*: En este modo, las consignas de velocidad (m/s) y dirección (grados) se envían a través de un mando a distancia a los controladores PID. La capa de seguridad funciona exactamente igual que en el modo *Control por ROS*.
- *Control Remoto No Seguro*: Este modo permite el control totalmente manual de la plataforma, sin capa de seguridad, e incluso permite desactivar los controladores PID, asignando directamente las tensiones o PWMs a los motores.
- *Parada de Emergencia*: En este modo, la plataforma se detiene, desactivando los motores y activando los frenos. CLEAR va direcatamente a este estado si alguno de los interruptores de emergencia (RC o de a bordo) está activado, también si se pierde la señal del mando a distancia o si no se reciben los *topics* de la capa

de seguridad reactiva (a menos que el modo operativo actual sea *Control Remoto No Seguro*). Para salir del modo de *Parada de Emergencia* se deben reajustar todos los interruptores de emergencia, el interruptor de modo de funcionamiento RC se debe poner en *Control Remoto No Seguro* y, por último, se debe girar la rueda de rearme/bocina hasta alcanzar la posición de rearme.

Interfaz con ROS

Para un correcto control del vehículo y depuración del código se ha incluido una interfaz de alto nivel con ROS, ya que este permite una comunicación robusta entre CLEAR y el exterior. La comunicación se ha basado en *topics* de ROS. Para ello se han definido 3 arquetipos de mensajes:

- *Mensajes de Control*: Estos *topics* permiten mandar al vehículo los estados Ackermann deseados (cómo pueden ser velocidad y dirección, entre otros). También permite mandar los valores a las variables que pueden ser dinámicas en la plataforma (como pueden ser ganancias de los controladores, valores para los filtros, etcétera). A través de un *topic* también podemos configurar el nivel de información que envía el vehículo para el depurado.
- *Mensajes de Monitorización*: Estos *topics* permiten visualizar las variables medidas o estimadas por CLEAR, como puede ser la velocidad o el ángulo de giro de la dirección, incluso la tensión que se le aplica a estos motores.
- *Mensajes de Depurado*: Estos *topics* permiten visualizar los valores de los comandos que le hemos enviado a través de los *topics de control* para asegurarnos de que internamente está funcionando correctamente.

Controladores y Estimadores del Estado

Para realizar un control robusto de la plataforma hay que ser capaz de medir de la forma más precisa posible la velocidad y dirección. Esto no es siempre posible debido

a la configuración previa de la plataforma, con en el caso de BLUE.

CLEAR está preparado para medir el estado usando codificadores incrementales (*encoders*). Esto hace que, dependiendo de la resolución del codificador, se tenga una mayor o menor precisión en la medida.

Debido a que la plataforma no tenía sensores para medir la velocidad, se diseñó un codificador incremental *"ad hoc"* impreso en 3D para acoplarlo al eje trasero. Este sensor, usa el principio de *Efecto Hall* para detectar la posición del eje. Por la baja resolución del sensor, se introdujo un Filtro de Kalman Unidimensional para estimar la velocidad [del Pino et al., 2018b]. Este filtro tienen la ventaja de ser extremadamente ligero computacionalmente.

Pero debido al carácter genérico de CLEAR, es posible que diversas plataformas puedan tener una baja resolución tanto en la velocidad como en la dirección. Por ello se ha introducido un Filtro de Kalman Extendido de 3 dimensiones, es decir, que realiza la estimación de 3 variables de estado: la velocidad, la variación del ángulo de giro y el ángulo de dirección inicial, ya que este puede ser desconocido (ver Figura 5.2).

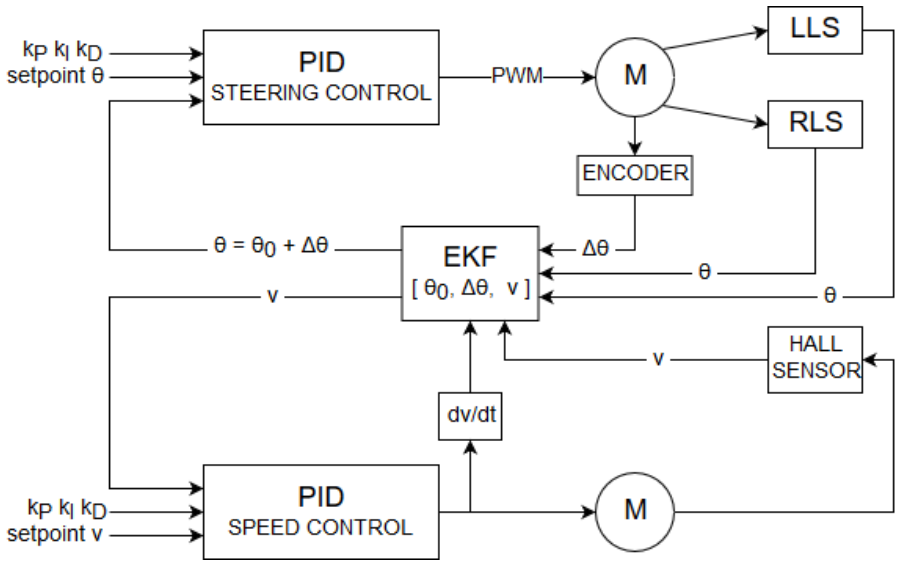


Figura 5.2: Esquema de control basado en controladores PID con un Filtro Kalman Extendido como estimador del estado.

Códigos de Error

Los códigos de error se utilizan para informar de los posibles errores o advertencias de seguridad a través de ROS.

En cuanto a los errores, se usan 6 tipos de error diferentes:

- No Errores.
- Control Remoto Perdido.
- Botón de Emergencia de CLEAR Activado.
- Botón de Emergencia del Mando Radio Control Activado.
- Mensaje de Control Reactivo No Recibido.
- Mensaje de Control de ROS No Recibido.

En cuanto a las advertencias de seguridad, se usan 5 tipos de advertencias diferentes:

- No Advertencias.
- Recibiendo Control por ROS Estando en Otro Modo.
- Limitación de la Velocidad por la Capa Reactiva.
- Tensión del Motor Fuera de Rano.
- Señal de Control Tratando de Exceder los Límites de la Dirección.

5.1.2. Hardware

Siguiendo la filosofía de hardware libre, los esquemas electrónicos de CLEAR se pueden descargar desde el repositorio [AUROVA, 2018].

El módulo principal de CLEAR es una placa Arduino Mega 2560. Esta ha sido elegida debido al mayor número de entradas y salidas, memoria, tamaño y precio frente a otros microcontroladores. Desde Arduino se realiza la comunicación con las interfaces

de alto nivel (ROS) y las de bajo nivel (sensores y actuadores). También incorpora un microcontrolador Teensy, para contabilizar la cuenta incremental de los codificadores incrementales de los motores, ya que minimiza la perdida de pulsos al estar totalmente dedicada a ello. Esta perdida puede no suponer un gran problema con codificadores de baja resolución, pero con altas resoluciones o reductoras, el error se incrementa y es necesario satisfacer a ambos tipos de sensores al querer realizar un módulo genérico.

CLEAR también incluye una interfaz visual con un LED RGB de alta luminosidad para poder visualizar el modo operacional de la plataforma en condiciones de luz ambiente.

Se ha desarrollado una PCB para interconectar todos los componentes necesarios dentro de CLEAR y así aislar los componentes y tensiones para mayor seguridad y facilidad de reemplazo. Además, para la interfaz con los sensores se han desarrollado otras placas de circuito impreso que incluyen acondicionamiento aislado para señales digitales de detectores y codificadores NPN o PNP, mediante optoacopladores.

CLEAR también incluye reguladores de tensión para funcionar con las propias baterías de la plataforma o externas. La tensión de entrada es de 24V, que sirve para alimentar los sensores y a todos los componentes del módulo.

5.1.3. Seguridad

Aunque ya han sido mencionadas algunas de las medidas de seguridad, CLEAR también incluye muchas otras, debido a la importancia de la seguridad en este campo.

Al estar embebido en un microprocesador, es necesario que CLEAR sea capaz de detectar si ha habido algún fallo que haga que el microprocesador no trabaje al tiempo de ciclo máximo permitido. Esto se implementa mediando un *watchdog* que reinicia el microcontrolador. Al iniciarse CLEAR entrará en modo de emergencia.

Pero también hay que tener en cuenta los rangos de las variables a controlar. Cualquier valor anómalo ha de ser desechado, esto se hace a través de los ficheros de configuración mencionados en este capítulo.

Cualquier corte en las medidas físicas de seguridad (como las setas de emergencia), perdida de la señal con el mando, o activación de alguno de los botones de emergencia, llevarán al vehículo al modo de EMERGENCIA, parando los motores por código y cortando la alimentación de forma física.

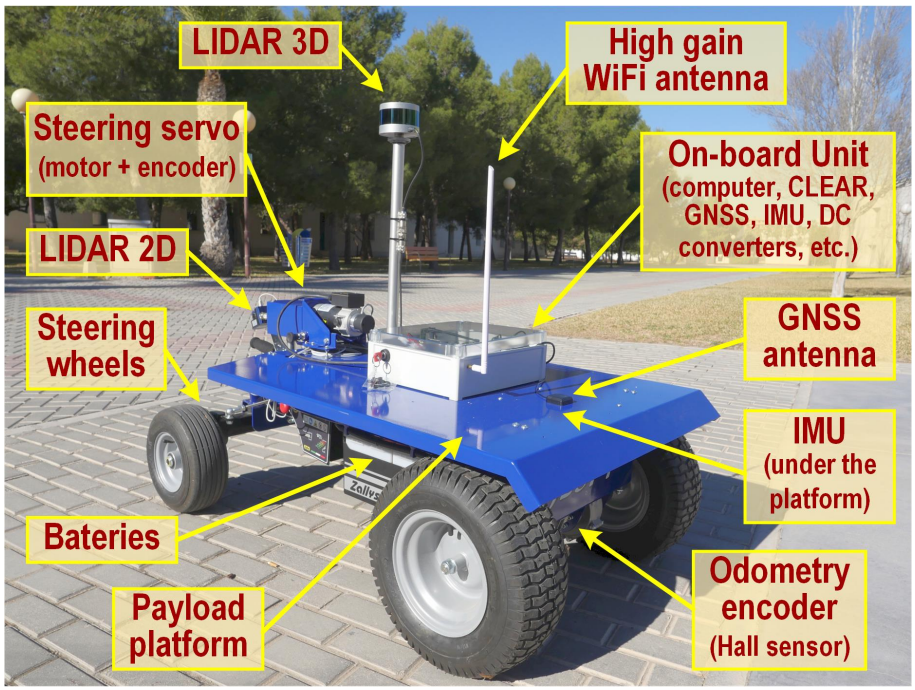


Figura 5.3: Robot BLUE con todos los componentes, incluyendo a CLEAR.

5.1.4. Integración en BLUE

CLEAR ha sido total y satisfactoriamente integrado en la plataforma industrial Jespy Zallys. Para ello se ha tenido que modificar la configuración inicial de la carretilla ya que no estaba diseñada para su automatización.

En cuanto a la dirección y la velocidad, se le ha añadido un motor de corriente continua (DC) y un *encoder* de alta resolución para controlar la dirección; y para la velocidad se ha utilizado el propio motor añadiendo un *encoder* en el eje trasero.

Mediante los ficheros de configuración se han definido los rangos de los motores y características propias de la plataforma, entre otras.

Finalmente, CLEAR ha sido integrado en la plataforma, renombrándola como BLUE (ver Figura 5.3). La cual está siendo utilizada hoy en día en investigación en robótica móvil gracias a CLEAR. Por tanto, se asume que la integración ha sido un éxito.

Capítulo 6

Experimentos

6.1. Percepción

En esta sección se describen los experimentos realizado en cuanto a percepción.

Debido a que este trabajo se ha centrado en el control, no se han realizado experimentos con redes de detección de objetos.

6.1.1. Segmentación

En cuanto a los experimentos con la segmentación de imagen basados en Deep Learning se han usado 3 arquitecturas distintas: **VggSegnet** [Badrinarayanan et al., 2017], **Xception** [Chollet, 2017] y **MobileNetV2** [Sandler et al., 2018].

Se han entrenado con un *dataset* extraído de CARLA con un total de 12908 imágenes, con distintas condiciones climáticas (soleado, lluvia, atardecer y nublado).

Los entrenamientos mostraron que **VggSegnet** era la más rápida en aprender, mientras que **MobileNetV2** y **Xception** tardaban mucho más y finalmente tenían resultados peores. Ambas, son redes más profundas, es por esto que para datos más simples (sintéticos) con pocas clases (13 en total) y al tener que utilizar *mini-batches* de 2

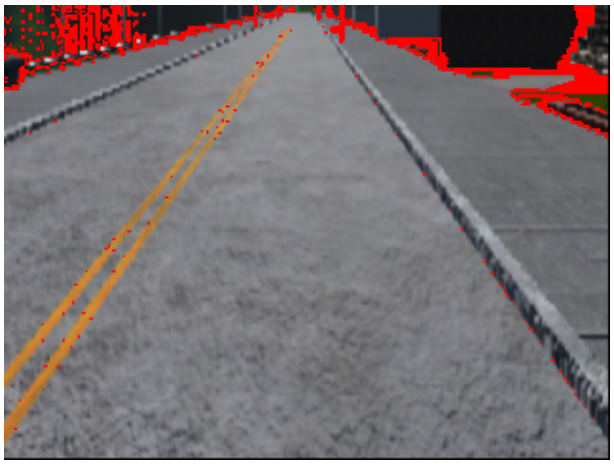


Figura 6.1: Comparación de la segmentación semántica con el pseudo-sensor del simulador CARLA y la salida del VggSegNet. En rojo, píxeles con etiquetado incorrecto. Exactitud del 95,04 %.

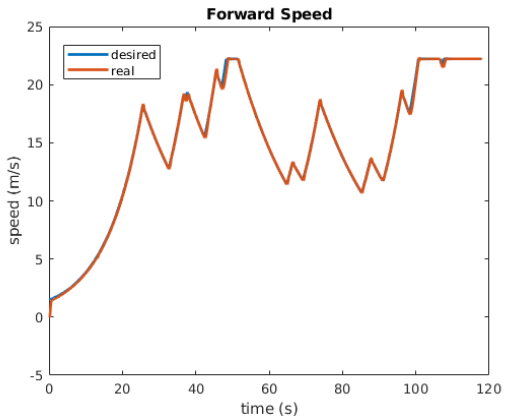
imágenes, VggSegnet preentrenada con los pesos de ImageNet tiene mejor resultado. En la Figura 6.1 se puede ver que después de 62 épocas, el error entre la segmentación propuesta por la red y CARLA es menor al 5 %.

6.2. Control

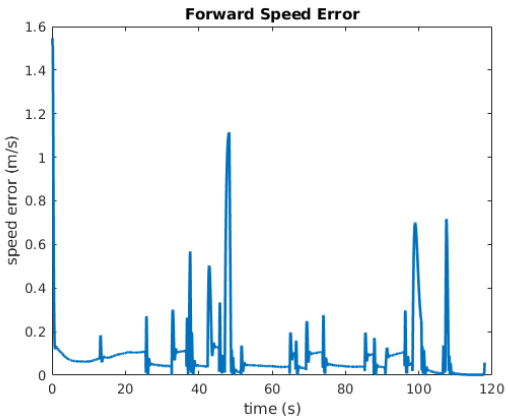
En esta sección se abordan los experimentos relacionados con los controladores utilizados. En primera instancia se han comparado los controladores clásicos Pure-Pursuit y Stanley debido a su grandes similitudes. Debido a su falta de robustez a la hora de fallos en la localización, se han utilizado otros controladores que utilizan información local de la cámara en vez de en la geometría de la posición.

Finalmente se ha realizado una comparación entre ambos métodos (basado en Imitation Learning (IL) y puramente reactivo basado en imagen). Para ello se han desarrollado unas métricas que determinen que agente funciona mejor.

Todos estos experimentos se han desarrollado usando el simulador CARLA.



(a) Comparación entre la velocidad deseada y la real



(b) Error en el tiempo de la velocidad

Figura 6.2: Resultados de aplicar un controlador PID a la velocidad.

6.2.1. Control Longitudinal

Para el control de velocidad se ha implementado usando un controlador PID que transforme el error en velocidad a una posición deseada del acelerador y del freno. Debido a la falta de conocimiento del mecanismo interno del vehículo no se ha podido implementar un controlador de bajo nivel.

Se ha probado este controlador en un rango de de velocidades de 0 a 80 km/h. Como se puede observar en la Figura 6.2, el controlador en velocidad es capaz de abarcar un amplio rango de velocidades con un error cuyo máximo se presenta puntualmente en 1.1 m/s.

Por tanto, se demuestra que no es necesario conocer la dinámica interna del vehículo para realizar un control de velocidad aceptable, ya que se puede hacer con un solo controlor PID. Pero, para un control mucho más preciso, es necesario realizar los dos controladores propuestos en el capítulo 4.

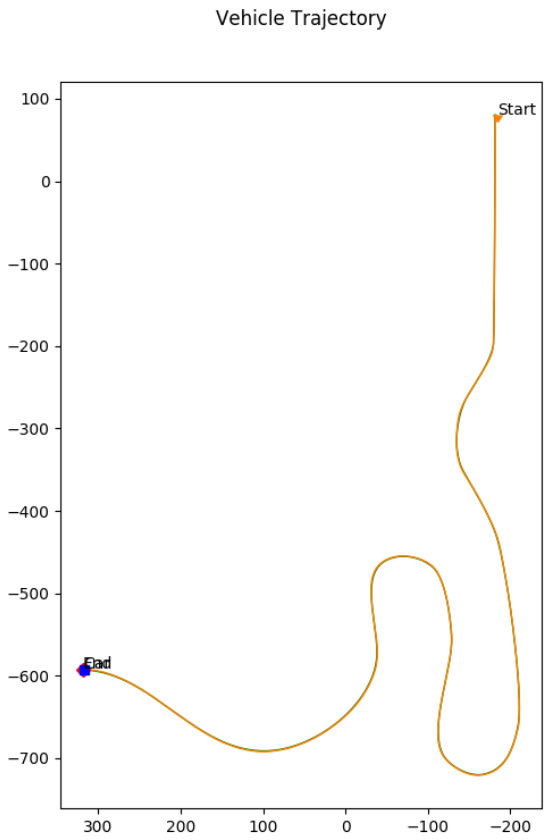


Figura 6.3: Trayectoria seguida por el vehículo.

6.2.2. Control Lateral Geométrico

En primer lugar se va a analizar el comportamiento de los algoritmos de control geométrico (Pure-Pursuit y Stanley) sobre una carretera al estilo autovía, con un carril amplio, con tramos muy rectos y curvas poco pronunciadas (ver Figura 6.3) que en total se corresponde con una distancia de 1.76 km que debe recorrer el vehículo.

En este tipo de situación, ambos controladores tienen un comportamiento similar, consiguen llegar al destino sin salirse de la carretera ni haciendo movimientos buscos del volante utilizando la posición de simulación sin errores. Pero esto no siempre es posible, los sensores introducen ruido en la medición y hacen que la localización no sea perfecta. Se ha simulado dicho ruido añadiendo ruido aleatorio a la localización

creciente en cada experimento para ver como afectaba al seguimiento de la trayectoria (ver Figura 6.4[1]). Los errores mayores a 1 metro hacen que la conducción sea imposible, no solo porque se salga de la carretera, si no porque el error se traduce en giros más bruscos en la dirección y por lo tanto, no se puede circular de forma segura ni cómoda.

Por ello, ambos controladores se han descartado para ser utilizados en un escenario complejo (como puede ser una cuidad) donde las oclusiones con los edificios pueden ocasionar la perdida de la localización.

Por estas razones, se ha decidido optar por otro tipo de controladores que no dependan de la localización global, si no, que puedan controlar el vehículo de forma reactiva a través de la información proporcionada por cámaras.

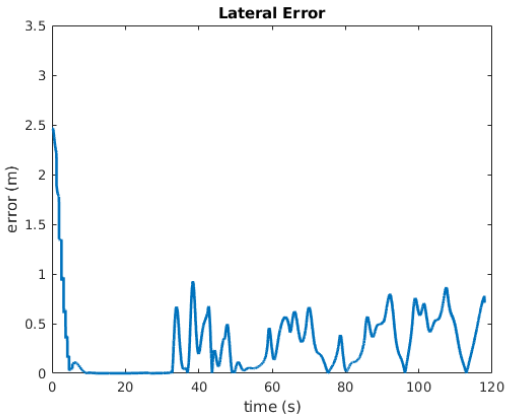
6.2.3. Control Lateral Basado en Cámaras

Los experimentos basados en control lateral basado en cámaras se han realizado probando 2 agentes, uno basado en Imitation Learning y otro basado control por imagen. Ambos utilizan comandos de alto nivel para saber qué hacer en la intersección como se menciona en el siguiente trabajo [Codevilla et al., 2017].

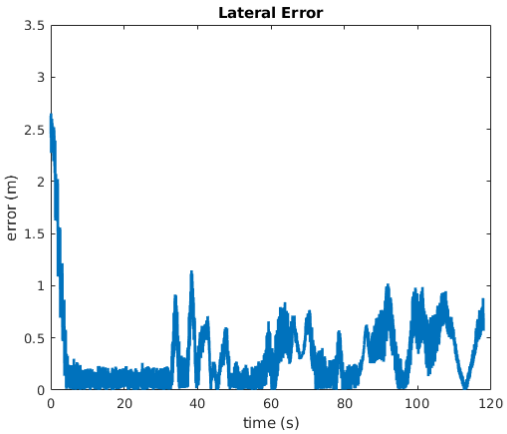
En cuanto al agente basado en IL, el primer enfoque fue entrenar diferentes arquitecturas para ver si tenían un comportamiento adecuado. Para ello se recabo una base de datos de más de 160.000 imágenes con sus correspondientes datos del vehículo (velocidad,dirección,comando) usando un mando de Playstation 3 para controlar el vehículo. El tiempo total de conducción para recabar la base de datos fueron de 3 horas. Una vez hecha la base de datos, se entrenaron diversas arquitecturas.

Se arquitecturas propuestas por otros investigadores [Kardell and Kuosku, 2017] [Commaai, 2016] [Bojarski et al., 2016]. Pero ninguna de ellas funcionó. Por ello se trató de aumentar la base de datos usando la utilizada por los desarrolladores de CARLA [CARLA, 2017]. Esto tampoco funcionó.

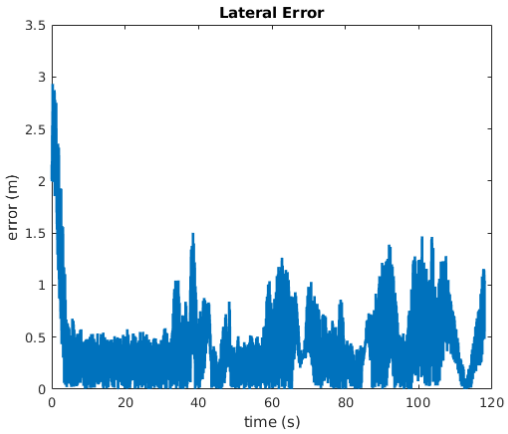
[1]Las figuras a), b), c), d) tiene limitado el eje Y hasta 3.5m, mientras que la e) esta limitada a 12m



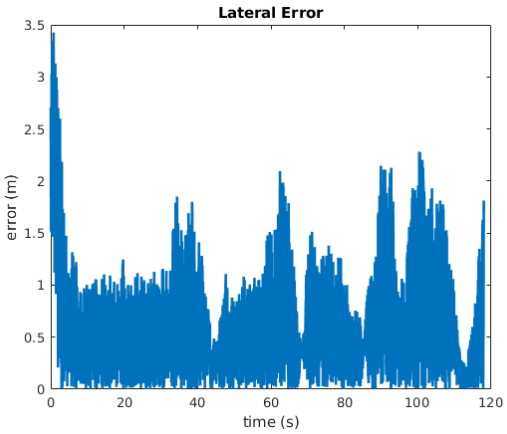
(a) Error de posición 0 m



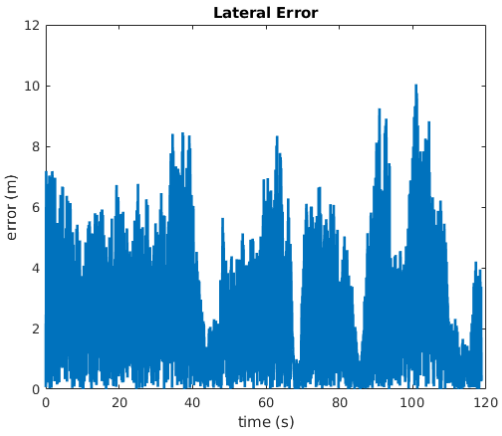
(b) Error de posición 0.2 m



(c) Error de posición 0.5 m



(d) Error de posición 1 m



(e) Error de posición 5 m

Figura 6.4: Error entre la posición deseada y la real.

Una vez vistos los infructuosos resultados se probó a hacer una red más compleja añadiendo diversas ramas que combinaban las arquitecturas anteriores y confluían en las capas de salida, además de añadir la información de velocidad, dirección y comando a estas capas. Después de días de entrenamiento ninguna de las redes fue capaz de proporcionar una respuesta adecuada.

Se cree que el fracaso de este entrenamiento puede ser debido a la limitación del hardware y a tener que utilizar *mini-batches* demasiado pequeños a la hora de entrenar el sistema.

Finalmente se utilizó la misma arquitectura propuesta por Codevilla et al. [Codevilla et al., 2017] para el simulador CARLA, una red distribuida en varias ramas, en la cual se utiliza una u otra dependiendo del comando de alto nivel.

Métricas

Una conducción humana confortable requiere unas condiciones muy específicas. Obviamente la seguridad es esencial, está prohibido chocar el coche o salir de su carril. Pero hay otras variables a considerar como la velocidad, la aceleración, el control de dirección, etc. Para lograr una mejor conducción se necesita una alta velocidad, pero en contraste, una alta aceleración tiene un efecto negativo. Lo mismo sucede con la dirección, se necesitan giros suaves del volante para que la conducción sea confortable y segura.

Por tanto, para comparar ambos métodos (basado en imagen e IL), se necesita una métrica representativa. En ese sentido, se ha recopilado toda la información sobre dirección, velocidad, tiempo (para obtener la derivada de primer orden de dirección y velocidad), y otra información útil como salirse del carril o de la carretera, colisiones, etc. Con todos los datos se calcula la media, los valores máximos, una fórmula de puntuación para cada variable y una puntuación global para medir cuán bueno o malo es cada método.

$$s_{speed} = \begin{cases} \frac{max_{speed} - |speed_i|}{max_{speed}} & \text{if } |speed_i| > mean_{speed} \\ -\frac{mean_{speed} - |speed_i|}{mean_{speed}} & \text{otherwise} \end{cases} \tag{6.1}$$

$$s_{accel} = \begin{cases} -\frac{max_{accel} - |accel_i|}{max_{accel}} & \text{if } |accel_i| > mean_{accel} \\ \frac{mean_{accel} - |accel_i|}{mean_{accel}} & \text{otherwise} \end{cases} \tag{6.2}$$

$$s_{\dot{steer}} = \begin{cases} -\frac{max_{\dot{steer}} - |\dot{steer}_i|}{max_{\dot{steer}}} & \text{if } |\dot{steer}_i| > mean_{\dot{steer}} \\ \frac{mean_{\dot{steer}} - |\dot{steer}_i|}{mean_{\dot{steer}}} & \text{otherwise} \end{cases} \tag{6.3}$$

La ecuación (6.3) utiliza la derivada de la dirección para calcular la puntuación (como la aceleración en el caso de la velocidad). Las métricas de medida: derivada de dirección, derivada de velocidad y velocidad toman valores normalizados entre -1 y 1 en cada instante. Pero las métricas críticas, siempre se calculan como un valor negativo. Teniendo en cuenta que se penaliza proporcionalmente en caso de que el vehículo se salga de su carril e invada el carril contrario.

Para comparar ambos métodos, un valor normalizado entre 0 y 1 podría ser suficiente. Pero si hay condiciones específicas, como por ejemplo una velocidad deseada, o una aceleración máxima, es más robusto usar valores negativos para descontar el comportamiento del agente cuando no alcanza los valores deseados.

Posteriormente, con las métricas (de medida y críticas) se calcula una puntuación global (6.4) (6.5).

$$S_{optimistic} = \frac{1}{nsamples} \sum_{i=1}^{nsamples} s_i \tag{6.4}$$

Donde:

$$s_i \rightarrow \begin{cases} -1 & \text{if collision} \\ -1 * intersection & \text{if intersection} \\ \frac{s_{speed}+s_{accel}+s_{steer}}{3} & \text{otherwise} \end{cases} \tag{6.5}$$

$$S_{real} = S_{opt} * \frac{n\_exp - failed\_exp}{n\_exp} \tag{6.6}$$

Para ser comparativos, la puntuación de cada variable se comparó con la mejor media y el valor máximo de ambos agentes. Como CARLA devuelve un valor entre 0 y 1 en la medida de intersección que es 0 si el coche está en el carril correcto y 1 si el coche está en el otro carril o fuera de la carretera, el descuento se calculó proporcionalmente al valor de la intersección. En el caso del valor de medida de colisión no está limitado, y como es lo peor que puede pasar, entonces siempre se descuenta el máximo. Tratando de ser lo más justo con ambos agentes, la puntuación de velocidad, aceleración y *dirección* tienen el mismo valor de peso para calcular la puntuación global (6.5).

Tenga en cuenta que la puntuación siempre es un número entre -1 y 1. Más tarde, este valor se interpola entre 0 y 1. Luego, la puntuación global da información de cuán buena o mala es la conducción del agente cuando el experimento tuvo éxito. Esto es lo que se llama la cota optimista. Sin embargo, para ser realista, el número de experimentos fallidos debe afectar el resultado final (6.6).

Resultados

Se han utilizado las métricas expuestas para determinar en qué casos el método de IL funciona mejor que el agente propuesto basado en imagen, para ello se han realizado los *benchmarks* propuestos en CARLA con ambos agentes. Cada experimento consiste en una posición inicial en el mapa y una posición de meta, el objetivo es alcanzar la meta lo más rápido posible. El experimento termina cuando el coche alcanza la posición de objetivo o cuando el tiempo excede el tiempo máximo del episodio. Un planificador de

rutas calcula el siguiente comando en cada paso para alimentar el sistema. Después de 74 experimentos y más de 27 kilómetros de conducción, los resultados se muestran en la Tabla 6.1, donde el color naranja indica que la métrica es peor que el otro agente y la marca roja la misma en el caso de métricas críticas de conducción.

Como se puede ver con sólo mirar los colores, el método por IL tiene un comportamiento peor en la mayor parte de los casos. Aun así, tan solo con los valores medios y máximos no se puede comprobar realmente cual es mejor o peor. Y la información temporal, como pueden ser las gráficas de velocidad, aceleración y derivada de la dirección son complicadas de interpretar ya que son muy similares.

Por lo tanto, el uso de las métricas descritas anteriormente es crucial. La Tabla 6.2 refleja los valores de las métricas de medición (6.1)(6.2)(6.3). La fórmula de la puntuación, utilizando los datos de todos los experimentos, reflejan que el método basado en imagen es más rápido y conduce de forma más suave que el método por IL. Además, la puntuación crítica muestra que es 70,51 veces mejor, en otras palabras, es más seguro. Los resultados revelan que la puntuación sobre las medidas y la puntuación optimista global es muy similar (6.4). Esto demuestra que, de hecho, la mayor parte del peso y efecto en la puntuación global es debido a las medidas, no a las colisiones ni a las intersecciones. Pero en el caso de la cota realista, hay una gran diferencia entre ambos métodos ya que el método basado en imagen pudo realizar todos los experimentos sin chocar, mientras que el método basado en IL falló 19 experimentos.

Tabla 6.1: Comparación cuantitativa.

Métricas[a]	Basado en Imagen	IL
tiempo medio	0.0019[b]	0.0120
velocidad máx	8.1488	10.4742
velocidad media	6.9507	5.3027
aceleración máx	14.0884	48.5741
aceleración media	0.3251	0.5239
dirección máx	18.8835	19.8093
dirección media	0.0234	0.1393
intersección off-road máx	0.0612	1
intersección off-road media	0.0052e-04	11.1861e-04
intersección carril máx	0.7142	0.8979
intersección carril media	3.2828e-4	13.1015e-04
colisión máx	0	6509.3681
colisión media	0	126.6847

[a] Velocidad y aceleración están en unidades SI. Mientras que *dirección* en $unidades/s$.
[b] Tiempo de segmentación (0.03s).

Tabla 6.2: Comparación de puntuaciones.

Puntuación	Basado en Imagen	IL
velocidad	0.7847	0.4027
aceleración	0.7459	0.4735
dirección	0.7233	0.5404
crítica	-1.1117e-04	-78.3870-04
optimista	0.7506	0.4622
realista	0.7506	0.3310

Capítulo 7

Conclusiones

En este trabajo se ha puesto de manifiesto la dificultad de la conducción autónoma en general así como la complejidad de cada tarea por separado. Por ello, este desarrollo se ha centrado en el control del vehículo, apoyándose en la percepción como pata fundamental para realizarlo.

Se ha puesto de manifiesto cómo los métodos tradicionales basados en controladores geométricos no son suficientemente robustos para la conducción autónoma. Por otro lado, los controladores más modernos basados puramente en *end-to-end learning* todavía están en fase de desarrollo por lo cual su fiabilidad no es del todo adecuada.

Por tanto, una combinación entre los métodos de control tradicionales, usando las ventajas de la inteligencia artificial para la percepción, son a día de hoy una de las técnicas más adecuadas para desarrollar la conducción autónoma, sobre todo cuando se trabaja con recursos (software/hardware) limitados.

Por otro lado, aunque los simuladores son necesarios para probar los algoritmos en una etapa temprana, también es necesario usar plataformas reales con las cuales probar los métodos desarrollados. Por ello, se ha desarrollado un módulo con la capacidad de controlar los motores a bajo nivel e informar del estado del vehículo. Pero probar con una plataforma real no siempre resulta fácil. Hay que tener en cuenta siempre la

perspectiva legal y es que para poder probar con un vehículo real se debe de hacer en un circuito cerrado o previa la autorización del organismo competente.

7.1. Trabajos Futuros

Como trabajo futuros, se quiere dar el salto de la prueba de estos algoritmos en un simulador a una plataforma real como puede ser BLUE. Además, se quiere profundizar en los conceptos de aprendizaje por refuerzo y aprendizaje profunda aplicados a la conducción autónoma (tanto en percepción como en visión) utilizando un equipo mejor.

Y por último, combinar todos estos conocimientos con los módulos de planificación y localización para poder desarrollar una aplicación real y funcional de conducción autónoma.

Apéndice A

Apéndice del Capítulo 4

En este Apéndice se aborda cómo modelar cinemática y dinámicamente un vehículo con geometría Ackermann (automóviles). Dependiendo de las restricciones geométricas del vehículo (o robot) se tendrá que aplicar un esquema u otro [Francis and Maggiore, 2016]. Hay que tener en cuenta que un buen modelo cinemático y dinámico es esencial para realizar el control sobre el vehículo.

Sin entrar en detalles, el modelo cinemático se relaciona con la posición y velocidad del vehículo, mientras que si se tienen en cuenta las fuerzas y momentos ejercidos, se está realizando un modelado dinámico. Como se discute en esta capítulo, el modelado cinemático funciona bien cuando el vehículo trabaja a velocidades bajas y la aceleración es insignificante, pero para tener un control fino del vehículo se debe realizar un modelado dinámico, sin embargo, este control es más complejo.

A.1. Modelado Cinemático

Para realizar el modelado cinemático de un coche autónomo, se usa un modelo simplificado llamado Modelo de Bicicleta. Este modelo simplifica la geometría Ackermann del vehículo en uno más sencillo respetando las restricciones no holonómicas. La rue-

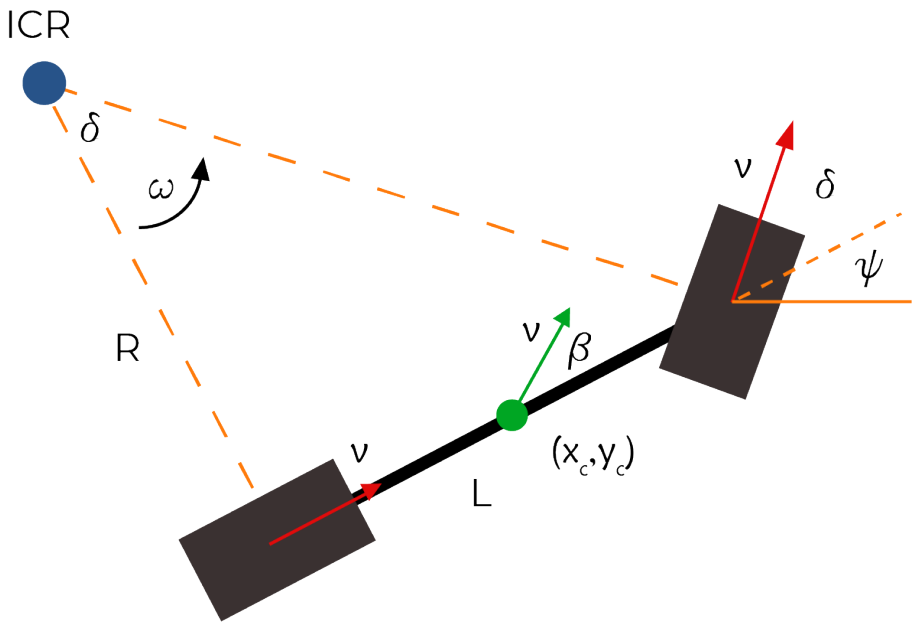


Figura A.1: Modelo cinemático de bicicleta usando el centro de gravedad como sistema de referencia.

da delantera se corresponderá con las dos ruedas directrices, y la rueda trasera con las ruedas motrices (para un vehículo de tracción trasera). Así, el ángulo que tome la rueda de bicicleta delantera se corresponderá con el ángulo de giro del vehículo y, por tanto, con la dirección de este.

Para obtener las ecuaciones matemáticas que describen el modelo, hay que tener en cuenta que sistema de referencia se va a utilizar, la rueda trasera, la rueda delantera o el centro de gravedad (ver Figura A.1).

Usando el Centro de Rotación Instantáneo (ICR):

$$\dot{\psi} = \omega = \frac{v}{R} \tag{A.1}$$

$$\tan\delta = \frac{L}{R} \tag{A.2}$$

En el caso de tomar como referencia el centro de gravedad:

$$\dot{x}_c = v \cdot \cos(\psi + \beta) \tag{A.3}$$

$$\dot{y}_c = v \cdot \sin(\psi + \beta) \tag{A.4}$$

$$\dot{\psi} = \frac{v \cdot \cos\beta \cdot \tan\delta}{L} \tag{A.5}$$

$$\dot{\delta} = \omega \tag{A.6}$$

$$\beta = \arctan\left(\frac{l_r \cdot \tan\delta}{L}\right) \tag{A.7}$$

Por tanto, este modelo es muy simple a la hora de implementarlo en código, ya que con la dirección y velocidades deseadas, integrando las ecuaciones del modelo se puede obtener la pose (x, y, ψ) del vehículo en un instante dado.

Código en Python:

```
delta = delta +  w * sample_time
beta = np.arctan(lr * np.tan(delta) / L)
theta = theta + (v * np.cos(beta) * np.tan(delta)/ L) * sample_time

xc = xc + (v * np.cos(beta + theta)) * sample_time
yc = yc + (v * np.sin(beta + theta)) * sample_time
```

A.2. Modelado Dinámico

El modelado dinámico del vehículo es importante debido a que cuando se desean altas velocidades o cuando hay deslizamiento en las ruedas (superficies resbaladizas) el modelo cinemático no funciona. Debido a que el vehículo se enfrenta a fuerzas y momentos de naturaleza distinta, hay que distinguir entre el modelo longitudinal y el lateral (velocidad y dirección respectivamente). Se asume, para la notación matemática de este trabajo, que el vehículo avanza hacia delante en el eje X, y que el eje Y corresponde a los cambios en dirección. Estos modelos se basan en la segunda ley de newton:

$$M \cdot a = \sum F \tag{A.8}$$

A.2.1. Modelado Longitudinal

A partir de la ecuación A.8 se puede determinar la ecuación de un vehículo determinando las fuerzas básicas que le pueden afectar cuando está avanzando de forma longitudinal.

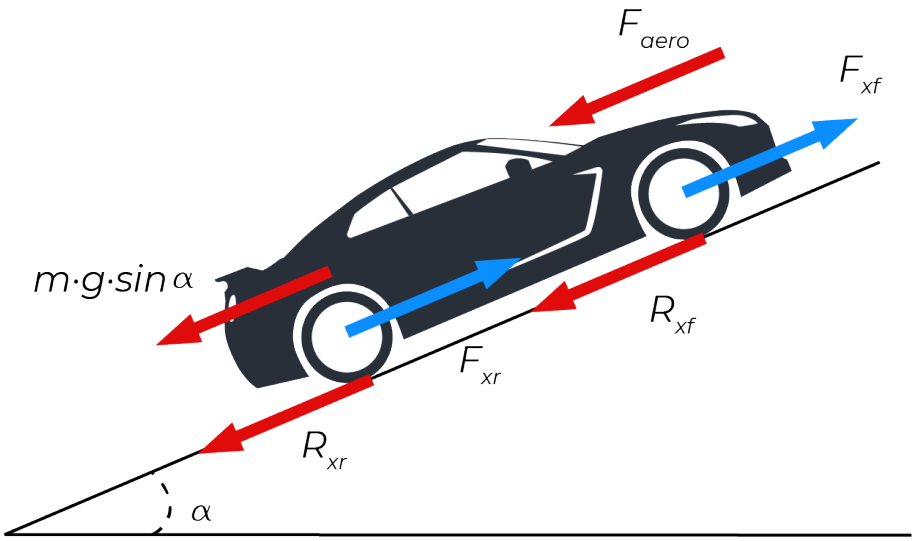


Figura A.2: Figura simplificada de las fuerzas que actúan sobre el vehículo de forma longitudinal.

$$m \cdot \ddot{x} = F_x - F_{aero} - R_x - m \cdot g \cdot \sin\alpha = F_x - F_{load} \tag{A.9}$$

$$\ddot{x} = r_{eff} \cdot GR \cdot \dot{\omega}_e \tag{A.10}$$

Sustituyendo y reordenando:

$$F_x = m \cdot \ddot{x} + F_{load} = m \cdot r_{eff} \cdot GR \cdot \dot{\omega}_e + F_{load} \tag{A.11}$$

Finalmente, se obtiene que el modelo dinámico simplificado del motor:

$$J_e \cdot \dot{\omega}_e = T_e - GR \cdot r_{reff} \cdot F_{load} = T_e - T_{load} \tag{A.12}$$

La derivación matemática de cómo se obtienen estas fuerzas, resistencias e inercias se sale del objetivo de este trabajo, estas ecuaciones están explicadas en el trabajo de Rajamani [Rajamani, 2012].

Donde GR son los ratios de los engranajes combinados,r_{ref} es el radio efectivo del neumático, J_e es la inercia. Por último T_e es el par deseado en el motor. Con estas ecuaciones se puede llegar a obtener la velocidad del vehículo a partir de una posición del acelerador y un ángulo de inclinación.

Código en Python:

```
x = x + v * sample_time
v = v + a * sample_time

w_e = w_e + w_e_dot * sample_time
w_w = GR * w_e
s = (w_w * r_e - v)/(v)

F_x = F_max
if abs(s) < 1:
    F_x = c * s

F_g = m * g * np.sin(alpha)
R_x = c_r1 * v
F_aero = c_a * v * v

F_load = F_aero + R_x + F_g
a = (F_x - F_load)/m

T_e = throttle*(a_0 + a_1 * w_e + a_2 * w_e * w_e)
w_e_dot = (T_e - GR * r_e * F_load)/J_e
```

A.2.2. Modelado Lateral

El modelo dinámico lateral de un vehículo que aquí se presenta está basado en el modelo cinemático de bicicleta. Se asume que la velocidad longitudinal es constante y que tanto la inclinación de la carretera como la aerodinámica es negligible.

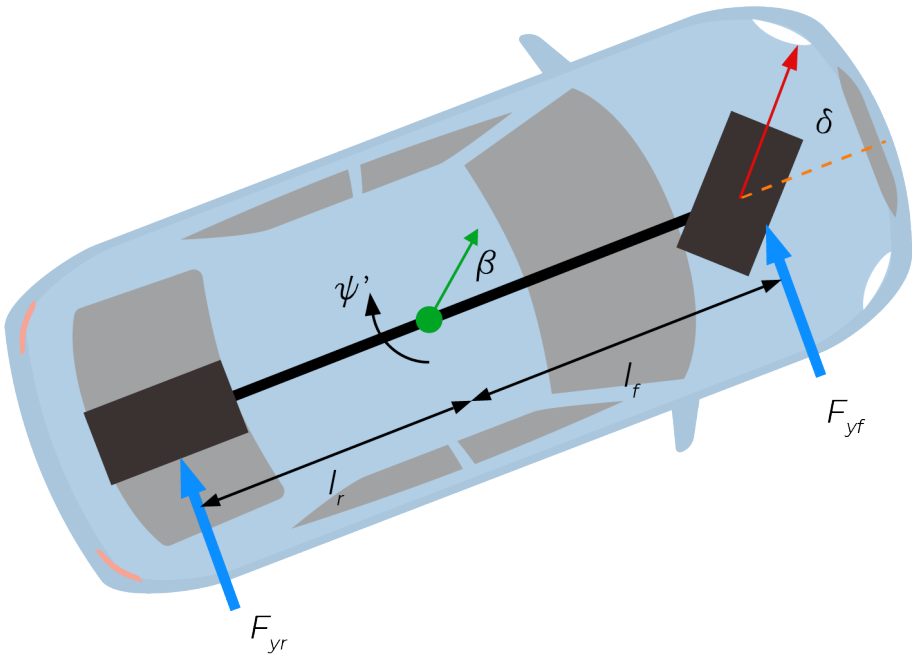


Figura A.3: Figura simplificada de las fuerzas que actúan sobre el vehículo de forma lateral.

Igual que en el modelado longitudinal, se usa la segunda ley de Newton para conocer las fuerzas que se ejercen sobre el vehículo.

$$m \cdot a_y = F_{yf} + F_{yr} \tag{A.13}$$

Por otro lado, la aceleración total del coche se puede calcular como:

$$a_y = \ddot{y} + \omega^2 \cdot R = V_x \cdot \dot{\beta} + V_x \cdot \dot{\psi} \tag{A.14}$$

Sustituyendo:

$$m \cdot V_x \cdot (\dot{\beta} + \dot{\psi}) = F_{yf} + F_{yr} \tag{A.15}$$

Los momentos sobre el eje Z se pueden calcular como:

$$I_z \cdot \ddot{\psi} = l_f \cdot F_{yf} + l_r \cdot F_{yr} \tag{A.16}$$

Sustituyendo y reorganizando las ecuaciones se pueden obtener los valores de $\dot{\beta}$ y $\ddot{\psi}$.

La derivación matemática de donde se obtienen estos valores se sale del objetivo de este trabajo, estas ecuaciones están explicadas en el trabajo de Rajamani [Rajamani, 2012].

A.3. Implementación Pure Pursuit

La implementación de Pure Pursuit se ha realizado teniendo en cuanta que los *goals* de la trayectoria están en coordenadas mundo, por tanto, hay que transformarlas a coordenadas del vehículo, como ocurre en el simulado CARLA. Una vez transformado a las coordenadas del robot, el cálculo del *Cross-Track Error* es trivial. Por otro lado para la ganancia del controlador se ha definido una velocidad máxima, mínima y una distancia mínima y máxima de búsqueda. Suponemos conocido el *yaw*, la longitud del vehículo, y la posición del robot.

Código en Python:

```
# Compute distance gain with v
k = 2.0 + (10.0 - 2.0) * (v - 0.0) / (25.0 - 0.0)

for idx in range(len(waypoints)):
    x_error = waypoints[idx][0] - x
    y_error = waypoints[idx][1] - y
```

```
    ld1 = np.sqrt((x_error*x_error)+(y_error*y_error))

    if ld1 >= k:
        break

# Change coordinate systems World to robot
p_W =np.array([waypoints[idx][0],waypoints[idx][1],0])

Rot_W2R = [[  np.cos(yaw) , np.sin(yaw), 0],
           [-1*np.sin(yaw), np.cos(yaw), 0],
           [             0,           0, 1]]

Rot_W2R = np.array(Rot_W2R)

R = np.array([x,y,yaw])
p_R = Rot_W2R.dot(p_W - R)

# Compute Cross-track error
x_error = p_R[0]
y_error = p_R[1]

# Compute controller
steer_output = np.arctan2(2.0 * L*y_error,ld1*ld1)
```

A.4. Implementación Stanley

De forma similar que en A.3, se hacen las mismas suposiciones.

Código en Python:

```
#K gain
k = 0.5

Rot_W2R = [[   np.cos(yaw), np.sin(yaw), 0],
           [-1*np.sin(yaw), np.cos(yaw), 0],
           [             0,           0, 1]]
Rot_W2R = np.array(Rot_W2R)

R = np.array([x,y,yaw])

p1_W = np.array([waypoints[0][0],waypoints[0][1],0])
p2_W = np.array([waypoints[1][0],waypoints[1][1],0])

p1_R = Rot_W2R.dot(p1_W - R)

x_error = p1_R[0]
y_error = p1_R[1]

yaw_waypoint = np.arctan2(p2_W[1]-p1_W[1],p2_W[0]-p1_W[0])

phi = yaw_waypoint - yaw

# Compute controller
steer_output = phi + np.arctan2(k*y_error,v)

# Steering limits
if steer_output > 1.22:
    steer_output = 1.22
```

```
elif steer_output < -1.22:
    steer_output = -1.22
```

Bibliografía

[AUROVA, 2018] AUROVA (2018). The clear repository. Available as: https://github.com/AUROVA-LAB/aurova_clear.

[Badrinarayanan et al., 2017] Badrinarayanan, V., Kendall, A., and Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495.

[Ben-Daya, 2009] Ben-Daya, M. (2009). *Failure Mode and Effect Analysis*, pages 75–90. Springer London, London.

[Bernhard Wymann, 2014] Bernhard Wymann, Eric Espié, C. G. C. D. R. C. A. S. (2014). TORCS, The Open Racing Car Simulator. http://www.torcs.org.

[Bojarski et al., 2016] Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., and Zieba, K. (2016). End to end learning for self-driving cars. *CoRR*, abs/1604.07316.

[Bussemaker, 2014] Bussemaker, K. J. (2014). Sensing requirements for an automated vehicle for highway and rural environments.

[CARLA, 2017] CARLA (2017). Carla's imitation learning agent and dataset.

[Chollet, 2017] Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Codevilla et al., 2017] Codevilla, F., Müller, M., Dosovitskiy, A., López, A., and Koltun, V. (2017). End-to-end driving via conditional imitation learning. *CoRR*, abs/1710.02410.

[Commaai, 2016] Commaai (2016). the comma.ai driving dataset. Available as: https://github.com/commaai/research.

[Cova-Rocamora et al., 2018] Cova-Rocamora, S., del Pino, I., Muñoz-Bañon, M. A., Contreras, M. Á., Candelas, F. A., and Torres, F. (2018). Clear. un módulo para la robotización de máquinas ackermann. In *Jornadas de Automática*.

[Czarnecki, 2018] Czarnecki, K. (2018). *Operational Design Domain for Automated Driving Systems - Taxonomy of Basic Terms*.

[Dalal and Triggs, 2005] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1.

[del Pino et al., 2018a] del Pino, I., Cova, S., Contreras, M. A., Candelas, F. A., and Torres, F. (2018a). Presenting blue: A robot for localization in unstructured environments. In *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 130–135.

[del Pino et al., 2018b] del Pino, I., Muñoz-Bañón, M. Á., Contreras, M. A., Cova-Rocamora, S., Candelas, F. A., and Medina, F. T. (2018b). Speed estimation for control of an unmanned ground vehicle using extremely low resolution sensors. In *Proceedings of the 15th International Conference on Informatics in Control, Automation and Robotics, ICINCO 2018 - Volume 1, Porto, Portugal, July 29-31, 2018.*, pages 216–223.

[del Pino et al., 2019] del Pino, I., Muñoz-Bañon, M. Á., Cova-Rocamora, S., Contreras, M. Á., Candelas, F. A., and Torres, F. (2019). Deeper in blue. *Journal of Intelligent & Robotic Systems*.

[Dosovitskiy et al., 2017] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16.

[Francis and Maggiore, 2016] Francis, B. A. and Maggiore, M. (2016). *Models of Mobile Robots in the Plane*, pages 7–23. Springer International Publishing, Cham.

[Hoffmann et al., 2007] Hoffmann, G. M., Tomlin, C. J., Montemerlo, M., and Thrun, S. (2007). Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing. In *2007 American Control Conference*, pages 2296–2301.

[Kalra and Paddock, 2016] Kalra, N. and Paddock, S. (2016). Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice*, 94:182–193.

[Kardell and Kuosku, 2017] Kardell, S. and Kuosku, M. (2017). Autonomous vehicle control via deep reinforcement learning. Master's thesis, CHALMERS UNIVERSITY OF TECHNOLOGY, Sweden.

[Kendall et al., 2018] Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J.-M., Lam, V.-D., Bewley, A., and Shah, A. (2018). Learning to drive in a day. *CoRR*, abs/1807.00412.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA. Curran Associates Inc.

[M. Snider, 2011] M. Snider, J. (2011). Automatic steering methods for autonomous automobile path tracking.

[Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783.

[Motors,] Motors, G. Gm safety. Available as: https://www.gm.com/content/dam/company/docs/us/en/gmcom/gmsafetyreport.pdf.

[Rajamani, 2006] Rajamani, R. (2006). *Introduction to Longitudinal Control*, pages 123–152. Springer US, Boston, MA.

[Rajamani, 2012] Rajamani, R. (2012). *Longitudinal Vehicle Dynamics*, pages 87–111. Springer US, Boston, MA.

[Reinholtz, 2007] Reinholtz, C. (2007). Darpa urban challenge technical paper.

[Ren et al., 2015] Ren, S., He, K., Girshick, R. B., and Sun, J. (2015). Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497.

[Richter et al., 2018] Richter, S. R., Hayder, Z., and Koltun, V. (2018). Playing for benchmarks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, volume 00, pages 2232–2241.

[Richter et al., 2016] Richter, S. R., Vineet, V., Roth, S., and Koltun, V. (2016). Playing for data: Ground truth from computer games. *CoRR*, abs/1608.02192.

[SAE, 2018] SAE (2018). Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. Available as: https://www.sae.org/standards/content/j3016_201806/.

[Sandler et al., 2018] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

[TensorFlow,] TensorFlow. Tensorflow object detection api. Available as: https://github.com/tensorflow/models/tree/master/research/object_detection.

[Tesla,] Tesla. Autopilot. Available as: https://www.tesla.com/es_ES/autopilot.

[Thrun et al., 2006] Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., et al. (2006). Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692.

[Udacity, 2016a] Udacity (2016a). Udacity's self-driving car dataset. Available as: https://github.com/udacity/self-driving-car/tree/master/datasets.

[Udacity, 2016b] Udacity (2016b). Udacity's self-driving car simulator. Available as: https://github.com/udacity/self-driving-car-sim.

[Viola and Jones, 2001] Viola, P. A. and Jones, M. (2001). Robust real-time object detection.

[Wang et al., 2016] Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2016). Sample efficient actor-critic with experience replay. *CoRR*, abs/1611.01224.

[Watkins and Dayan, 1992] Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

[Waymo,] Waymo. Waymo safety. Available as: https://waymo.com/safety/.

[Wendorff, 2017] Wendorff, W. (2017). Quantitative sotif analysis for highly automated driving systems.

[Yang et al., 2018] Yang, Z., Zhang, Y., Yu, J., Cai, J., and Luo, J. (2018). End-to-end multi-modal multi-task vehicle control for self-driving cars with visual perception. *CoRR*, abs/1801.06734.

[Yu et al., 2016] Yu, A., Palefsky-Smith, R., and Bedi, R. (2016). Deep reinforcement learning for simulated autonomous vehicle control.